Universidad del País Vasco Euskal Herriko Unibertsitatea
BILBOKO INGENIARITZA ESKOLA
ESCUELA DE INGENIERÍA DE BILBAO

GRADO EN INGENIERÍA INFORMÁTICA DE GESTIÓN Y SISTEMAS DE INFORMACIÓN

TRABAJO FIN DE GRADO

EVALUACIÓN DE RENDIMIENTO DE LA PLATAFORMA DE COMPUTACIÓN CUÁNTICA IBM

Alumno: Echevarria, Unibaso, Lander

Director: Lopez, Novoa, Unai

Curso: 2021-2022

Fecha: 03, 02, 2022

Resumen

La computación cuántica es un área de la informática que actualmente se sigue estudiando y desarrollando pero que promete grandes cambios. Esto es debido a que teóricamente estos sistemas son capaces de almacenar muchos más estados de información que en la computación clásica y operar con algoritmos más eficientes.

Este proyecto se centra en probar distintas aplicaciones para comprobar si realmente se consiguen los resultados esperados. Para ello, en primer lugar, se estudia la base teórica de la computación cuántica y las aplicaciones en las que se puede utilizar.

Posteriormente, se analizan los entornos o infraestructuras que las empresas actuales ponen a disposición de los usuarios a través de la nube para poder trabajar con esta tecnología. De los sistemas analizados, tras decidir trabajar con la plataforma online IBM Q de la empresa IBM, se hace un análisis más profundo de él observando los sistemas que ofrece, sus funcionalidades y su kit de desarrollo de software Qiskit.

Utilizando las herramientas ofrecidas por IBM Q y Qiskit, se prueban varias aplicaciones: un algoritmo cuántico que promete ofrecer una aceleración cuadrática respecto a un algoritmo clásico, y aplicaciones en las áreas del aprendizaje automático, optimización de problemas y finanzas.

Finalmente, se analizan las pruebas realizadas para observar la usabilidad de las aplicaciones, los resultados obtenidos y tiempos de ejecución, los cuales indican que los computadores cuánticos reales de la plataforma IBM proveen tiempos de ejecución significativamente mayores que sus simuladores. También se proponen varias mejoras o ampliaciones que podrían realizarse en trabajos futuros.

Laburpena

Konputazio kuantikoa gaur egun aztertzen eta garatzen jarraitzen den baina aldaketa handiak agintzen dituen informatikaren arlo bat da. Izan ere, teorikoki sistema hauek konputazio arruntean baino askoz informazio egoera gehiago gordetzeko eta algoritmo eraginkorragoekin jarduteko gai dira.

Honenbestez, proiektu honen helburua espero diren emaitzak benetan lortzen diren egiaztatzeko aplikazio desberdinekin probak egitea da. Horretarako, lehenik eta behin, konputazio kuantikoaren oinarri teorikoak eta zein aplikaziotan erabil daitekeen aztertzen da.

Ondoren, gaur egungo enpresek hodei konputazioaren bidez teknologia honekin lan egin ahal izateko erabiltzaileen eskura jartzen dituzten inguruneak edo azpiegiturak aztertzen dira. Aztertutako sistemetatik, IBM enpresaren IBM Q online plataformarekin lan egitea erabaki ondoren, sistema honen azterketa sakonagoa egiten da, eskaintzen dituen sistemak, funtzionaltasunak eta Qiskit izeneko software grapen-kita behatuz.

IBM Qk eta Qiskitek eskainitako tresnak erabiliz, hainbat aplikazio probatuko dira: algoritmo arrunt batekin konparatuz denboran zehar hobekuntza koadratikoa eskaintzen duen algoritmo kuantiko bat eta ikasketa automatiko, optimizazio arazo eta finantzen arloetako aplikazioak.

Azkenik, aplikazioen erabilgarritasuna, lortutako emaitzak eta gauzatze-denborak behatzeko egindako probak aztertzen dira. Proba hauek, IBM plataformako benetako ordenagailu kuantikoek, simulagailuek baino exekuzio-denbora handiagoak dituztela adierazten dute. Horrez gain, etorkizunerako lanetan egin litezkeen hainbat hobekuntza edo gehikuntza ere proposatzen dira.

Abstract

Quantum computing is an area of computing that is currently still being studied and developed but one which promises great advances and innovations. This is because these systems are theoretically capable of storing many more states of information than their classic counterparts and of operating with much more efficiently with certain types of algorithms.

This project focuses on testing different applications to see if the expected results are achieved. To this end, firstly, theoretical basis of quantum computing and some of its case study applications are studied.

After that, the environments or infrastructures that current companies make available to users through cloud computing environments are analyzed. Of these systems, after deciding to work with the IBM Q online platform, a more in-depth analysis is made of it by observing the systems it offers, its functionalities and its Qiskit software development kit.

Using the tools offered by IBM Q and Qiskit, several applications are tested: a quantum algorithm that promises to offer a quadratic speedup over a classical algorithm and some applications from the areas of machine learning, problem optimization and finance.

Finally, tests are conducted to assess the performance and usability of the applications, the results obtained and execution times, which indicates that real quantum computers of the IBM platform provide significantly higher execution times than their simulators. Several improvements or extensions that could be made in future work are also proposed.

Índice de contenidos

Índice de figuras

Índice de tablas

1. INTRODUCCIÓN

Todos los días utilizamos sistemas informáticos, desde dispositivos móviles hasta ordenadores en nuestras casas o en empresas. La función principal de estos sistemas como los ordenadores o las computadoras es procesar grandes cantidades de datos de forma veloz y precisa, permitiéndonos resolver problemas que de otra forma no podríamos en un tiempo razonable.

Además, gracias a tecnologías como el aprendizaje automático o la inteligencia artificial, estos sistemas pueden llegar a obtener cierta capacidad de aprendizaje sin ser programados explícitamente para que puedan cambiar y actuar de forma más autónoma cuando se exponen a nuevos datos. Esto ha causado que hoy en día se utilicen también en áreas importantes como la medicina, con el análisis de millones de datos clínicos y proporcionando ayuda a la hora de hacer pronósticos o diagnósticos [1].

Sin embargo, este tipo de computación (también llamado computación clásica), tiene sus limitaciones, sobre todo cuando se trabaja con cantidades de datos superiores a los que puede manejar el sistema o almacenar en memoria y su poder de procesamiento no permite resolver los problemas o necesitaría de unos tiempos excesivamente grandes.

Para solucionar estos problemas, existe otro tipo de computación llamado computación cuántica, una rama de la informática que aprovecha los fenómenos de la mecánica cuántica como la superposición o el entrelazamiento. A causa de ello, teóricamente estas computadoras son capaces de almacenar y trabajar con muchos más estados de información que en la computación clásica y operar con algoritmos más eficientes [2].

La computación cuántica, cuyo origen remonta a 1981 cuando el físico estadounidense Paul Benioff expuso su teoría para aprovechar las leyes cuánticas en el entorno de la computación [3] , ha seguido desarrollándose hasta hoy y es un tema de investigación que sigue en activo con la esperanza de conseguir sobrepasar los límites actuales o mejorar las aplicaciones que ya conocemos.

Tal es el caso, que hoy en día la mayoría hemos escuchado mencionar algo al respecto sobre la computación cuántica, y grandes empresas como IBM, Amazon o incluso Google invierten en esta tecnología para explorar el potencial del cómputo cuántico [4]. Además, algunas de estas empresas están poniendo a disposición de los usuarios plataformas o entornos online con los que poder trabajar con esta tecnología.

Este proyecto es un trabajo de investigación para comprobar en qué estado se encuentran estos sistemas cuánticos y comprobar si realmente pueden mejorar los sistemas clásicos que ya conocemos o si en un futuro podrían llegar a sustituirlos.

Para ello, en este trabajo se realiza un estudio de los conceptos básicos de la computación cuántica junto con las aplicaciones en las que se puede utilizar y se realiza una exploración de la madurez de estas tecnologías que actualmente son usables a través de plataformas de cómputo en la nube como IBM Q, probando a resolver varios problemas en áreas como el aprendizaje automático o la optimización.

2. PLANTEAMIENTO INICIAL

En esta sección se detallan los objetivos que tiene que cumplir el proyecto y se define su alcance mediante la estructura de descomposición de trabajo (EDT). También se realiza una planificación del tiempo por cada tarea y se crea un diagrama de Gantt para especificar las fechas de cada uno. Por último, se realiza un análisis de riesgos y la evaluación económica.

2.1. Objetivos y captura de requisitos

Uno de los primeros objetivos de este trabajo es aprender los conceptos básicos de la computación cuántica y las áreas o aplicaciones en las que se puede utilizar, además de aprender la teoría o la funcionalidad general de estas aplicaciones para posteriormente poder experimentar con ellas.

Una vez adquirido un conocimiento base, otro de los objetivos es revisar qué proveedores existen y cuáles son los sistemas actuales que utilizan estas herramientas cuánticas. Para ello, se van a explorar sus capacidades, funcionalidades y entornos de trabajo.

Una vez revisados estos sistemas, el siguiente objetivo será seleccionar al menos uno con el que poder trabajar y empezar a experimentar con él para familiarizarse con sus funcionalidades y capacidades.

Por último, el objetivo principal por el que se desarrolla el trabajo es probar varios programas o aplicaciones en las áreas que se puede utilizar la computación cuántica y permita el sistema seleccionado. El resultado de ello será comprobar el rendimiento de ese sistema junto con una percepción de la usabilidad del mismo. Las aplicaciones con los que se va a experimentar son los siguientes: un algoritmo cuántico que ofrezca una mejora respecto a una versión clásica, una aplicación en el área del aprendizaje automático, otra aplicación en el área de la optimización y una última aplicación en el área de las finanzas.

Al inicio del proyecto se consideraban 2 alternativas para evaluar las capacidades reales de los sistemas de computación cuántica actuales:

- Probar la ejecución de al menos 2 aplicaciones en un sistema.
- Probar la ejecución de una aplicación en 2 sistemas distintos y hacer una comparativa de ellos.

De las opciones posibles, se ha decidido seguir la primera, es decir, seleccionar un único sistema con el que trabajar y probar en él varias aplicaciones. Por lo tanto, para completar el trabajo los objetivos que se tienen que cumplir son los siguientes:

- Aprender los conceptos básicos de la computación cuántica.
- Revisar los sistemas actuales y seleccionar uno de ellos para trabajar con él.
- Experimentar con el sistema seleccionado implementando distintas aplicaciones para observar el rendimiento y la usabilidad.

2.2. Alcance del proyecto

Para definir el alcance del proyecto, se ha realizado el diagrama EDT que estructura el trabajo en 4 bloques principales (ver Figura 2.1).

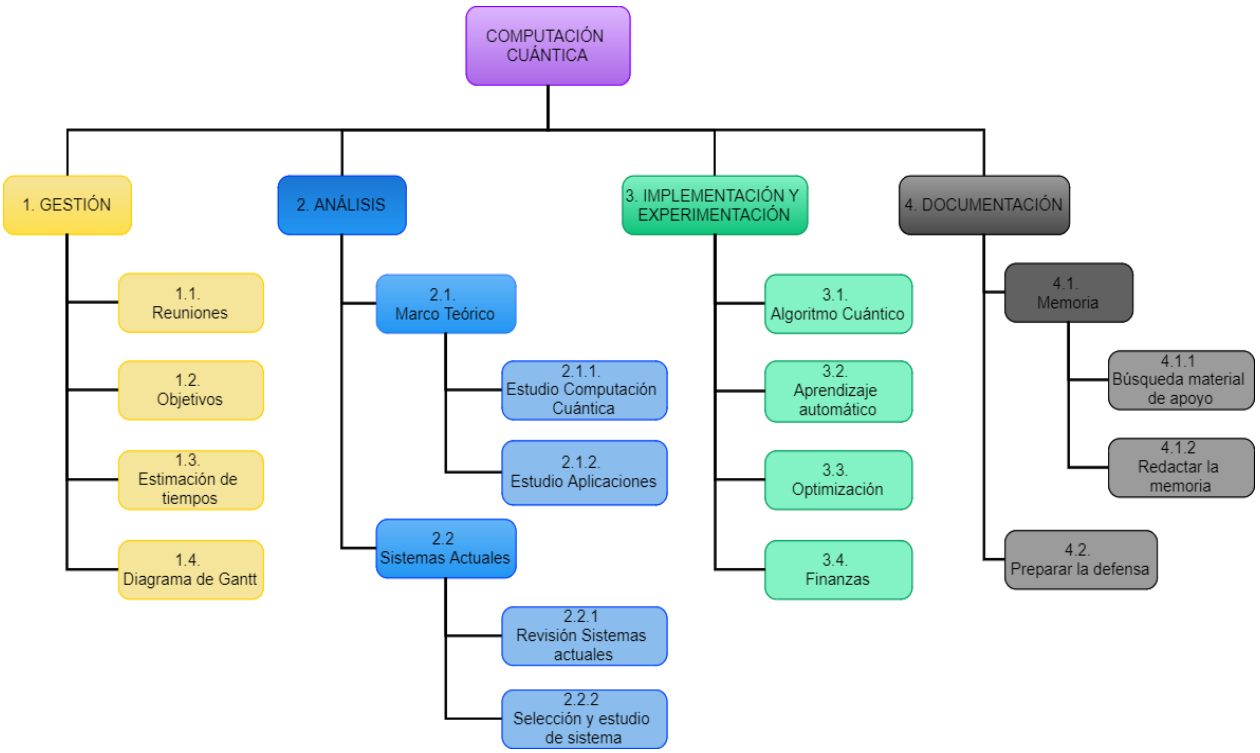


Figura 2.1 - Diagrama EDT

El primer bloque de **gestión** equivale a la primera fase del proyecto, aunque también contiene tareas que se alargan a lo largo del proyecto como las reuniones. Dentro de este bloque se encuentran las tareas para definir los objetivos del trabajo y la estimación de tiempos para cumplir cada tarea.

El segundo bloque de **análisis** equivale a la segunda fase del proyecto. En esta fase, se realiza un estudio de los conceptos básicos de la computación cuántica y a su vez se analizan las aplicaciones en las que se puede utilizar y se estudia la teoría básica de ellos. Por otro lado, se analizan los sistemas actuales que permiten a los usuarios utilizar este tipo de tecnología y después de seleccionar uno de ellos se hace un análisis más exhaustivo del mismo.

Una vez obtenido la base teórica necesaria y haber seleccionado un sistema con el que trabajar, en la siguiente fase de **implementación y experimentación** se trabaja en desarrollar y probar varios programas en áreas como el aprendizaje automático o la optimización para explorar las capacidades del sistema.

Por último, se encuentra la fase final del proyecto con el bloque de **documentación.** En esta fase se encuentran las tareas para redactar la memoria con la información y los datos obtenidos a lo largo de todo el proyecto junto con la búsqueda de información adicional para corroborar los resultados obtenidos y la preparación para la defensa del trabajo.

En las siguientes secciones se analiza cada bloque individualmente para definir más concretamente las tareas que componen cada uno. Además, se describirá cada tarea indicando el paquete de trabajo al que pertenece, el tiempo estimado para completarlo, la descripción y las entregas o salidas que genera.

2.2.1. Gestión

En este apartado se muestran las tareas del bloque de gestión que se irán realizando a lo largo del proyecto (ver Figura 2.2) y se detallará más información de cada uno (ver Tablas 2.1, 2.2, 2.3 y 2.4)

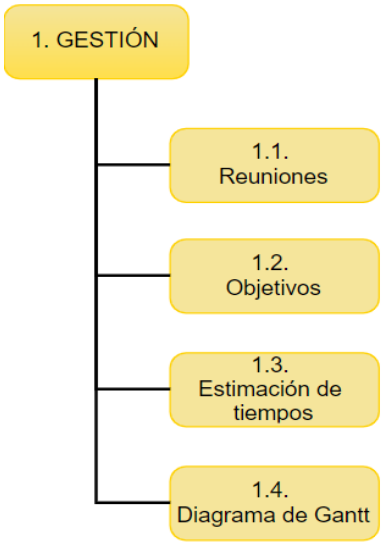


Figura 2.2 - Bloque de Gestión del diagrama EDT

1.1 Reuniones
Paquete de Trabajo: Gestión
Tiempo estimado: 15
Descripción: Reuniones con el tutor del proyecto para el seguimiento del trabajo. El objetivo de estas reuniones será resolver dudas o problemas, analizar el trabajo realizado y actualizar los objetivos en caso de ser necesario.
Entregas/Salidas: Anotaciones realizadas en las reuniones**.**
Recursos: Webex

Tabla 2.1 - Tarea Reuniones

1.2 Objetivos
Paquete de Trabajo: Gestión
Esfuerzo estimado: 4
Descripción: Fijar los objetivos y las tareas a realizar junto con los requisitos que debe de cumplir el proyecto.
Entregas/Salidas: Apartado de objetivos de la memoria.
Recursos empleados: Microsoft Word

Tabla 2.2 - Tarea Objetivos

1.3 Estimación de tiempos
Paquete de Trabajo: Gestión
Esfuerzo estimado: 3
Descripción: Estimar la duración para completar las tareas que componen el proyecto.
Entregas/Salidas: Apartado alcance del proyecto de la memoria.
Recursos empleados: Microsoft Word

Tabla 2.3 - Tarea Estimación de tiempos

1.4 Diagrama de Gantt
Paquete de Trabajo: Gestión
Esfuerzo estimado: 8
Descripción: Elaborar el diagrama de Gantt para definir el inicio de cada tarea junto con la duración y la fecha de finalización.
Entregas/Salidas: Diagrama de Gantt
Recursos empleados: GanttProject

Tabla 2.4 - Tarea Diagrama de Gantt

2.2.2. Análisis

En esta sección se muestran las tareas del bloque de análisis (ver Figura 2.3) para el estudio teórico de la computación cuántica y sus aplicaciones junto el análisis de los sistemas actuales que permiten utilizarlos. Se detalla la información de cada tarea en las Tablas 2.5, 2.6, 2.7 y 2.8.

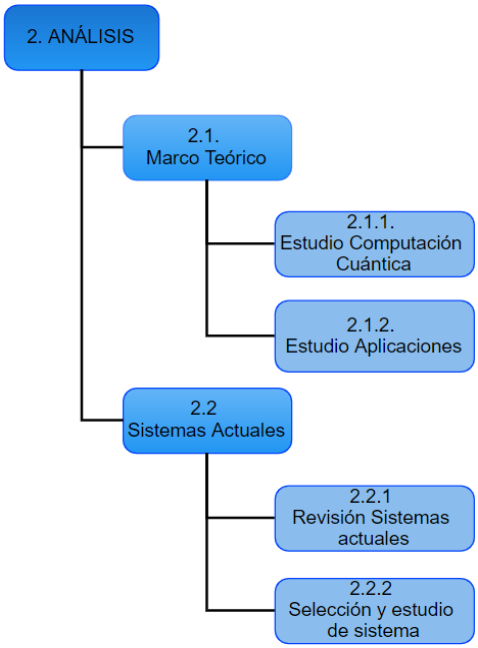


Figura 2.3 - Bloque de Análisis del diagrama EDT

2.1.1 Estudio computación cuántica
Paquete de Trabajo: Análisis
Esfuerzo estimado: 35
Descripción: Aprender los conceptos básicos y la teoría general sobre la computación cuántica.
Entregas/Salidas: Apartado Teoría Computación Cuántica de la memoria.
Recursos empleados: Fuentes online de documentación (incluyendo manuales IBM), Microsoft Word

Tabla 2.5 - Tarea Estudio computación cuántica

2.1.2 Estudio aplicaciones
Paquete de Trabajo: Análisis
Esfuerzo estimado: 40
Descripción: Estudiar las aplicaciones o las áreas en las que se puede utilizar la computación cuántica y aprender los conceptos y la teoría necesaria de un algoritmo cuántico y de las aplicaciones en las áreas del aprendizaje automático, la optimización y las finanzas.
Entregas/Salidas: Apartado Aplicaciones de la memoria.
Recursos empleados: Fuentes online de documentación (incluyendo manuales IBM), Microsoft Word

Tabla 2.6 - Tarea estudio aplicaciones

2.1.2 Revisión sistemas actuales
Paquete de Trabajo: Análisis
Esfuerzo estimado: 15
Descripción: Explorar proveedores y sistemas actuales que ofrecen servicios de computación cuántica analizando sus capacidades, recursos disponibles, entornos de programación, funcionalidades…
Entregas/Salidas: Apartado Sistemas Actuales de la memoria.
Recursos empleados: Fuentes online de documentación (incluyendo manuales IBM), Microsoft Word

Tabla 2.7 - Tarea Revisión sistemas actuales

2.1.2 Selección y estudio de sistema
Paquete de Trabajo: Análisis
Esfuerzo estimado: 20
Descripción: Seleccionar uno de los sistemas explorados en la tarea anterior y hacer un análisis más exhaustivo en cuanto a las capacidades, funcionalidades y su entorno de programación.
Entregas/Salidas: Apartado Sistemas Actuales de la memoria.
Recursos empleados: Fuentes online de documentación (incluyendo manuales IBM), Microsoft Word

Tabla 2.8 - Tarea Selección y estudio de sistema

2.2.3. Implementación y experimentación

En esta sección se muestran las tareas del bloque de implementación y experimentación (ver Figura 2.4) para probar las distintas aplicaciones que han sido estudiadas en el bloque anterior. Se detalla la información de cada tarea en las Tablas 2.9, 2.10, 2.11 y 2.12.

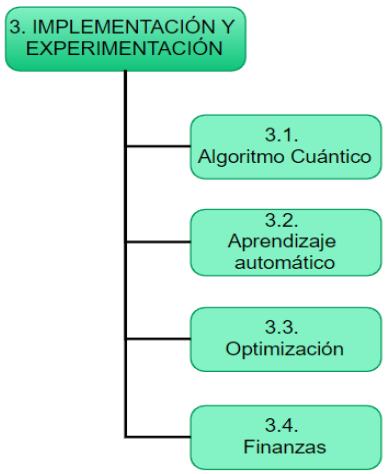


Figura 2.4 - Bloque de Implementación y Experimentación del diagrama EDT

3.1 Algoritmo Cuántico
Paquete de Trabajo: Implementación y experimentación
Esfuerzo estimado: 20
Descripción: Implementar un algoritmo cuántico en el sistema seleccionado en la fase anterior, probando distintas ejecuciones para obtener información en cuanto a los resultados, tiempos de ejecución y usabilidad.
Entregas/Salidas: Apartado Experimentación de la memoria
Recursos empleados: Jupyter Notebook, Python, sistema seleccionado, Microsoft Word

Tabla 2.9 - Tarea Algoritmo cuántico

3.2 Aprendizaje Automático
Paquete de Trabajo: Implementación y experimentación
Esfuerzo estimado: 20
Descripción: Experimentar con una aplicación en el ámbito del aprendizaje automático en el sistema seleccionado en la fase anterior, probando distintas ejecuciones para obtener información en cuanto a los resultados, tiempos de ejecución y usabilidad.
Entregas/Salidas: Apartado Experimentación de la memoria
Recursos empleados: Jupyter Notebook, Python, sistema seleccionado, Microsoft Word

Tabla 2.10 - Tarea Aprendizaje automático

3.3 Optimización
Paquete de Trabajo: Implementación y experimentación
Esfuerzo estimado: 20
Descripción: Experimentar con una aplicación en el ámbito de la Optimización en el sistema seleccionado en la fase anterior, probando distintas ejecuciones para obtener información en cuanto a los resultados, tiempos de ejecución y usabilidad.
Entregas/Salidas: Apartado Experimentación de la memoria
Recursos empleados: Jupyter Notebook, Python, sistema seleccionado, Microsoft Word

Tabla 2.11 - Tarea Optimización

3.4 Finanzas
Paquete de Trabajo: Implementación y experimentación
Esfuerzo estimado: 20
Descripción: Experimentar con una aplicación en el ámbito de las Finanzas en el sistema seleccionado en la fase anterior, probando distintas ejecuciones para obtener información en cuanto a los resultados, tiempos de ejecución y usabilidad.
Entregas/Salidas: Apartado Experimentación de la memoria
Recursos empleados: Jupyter Notebook, Python, sistema seleccionado, Microsoft Word

Tabla 2.12 - Tarea Finanzas

2.2.4. Documentación

En esta sección se muestran las tareas del bloque de documentación (ver Figura 2.5). Se detalla la información de cada tarea en las Tablas 2.13, 2.14 y 2.15

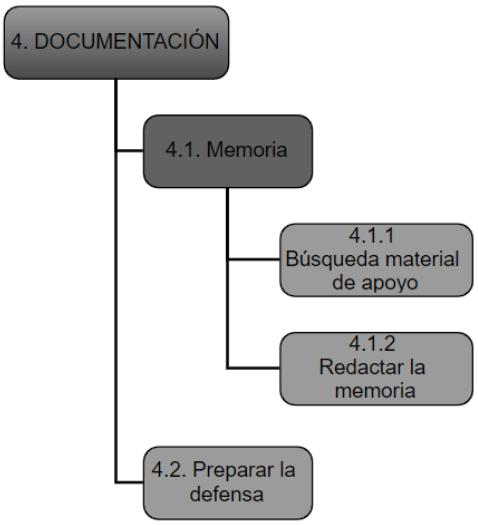


Figura 2.5 - Bloque de Documentación del diagrama EDT

4.1.1 Búsqueda material de apoyo
Paquete de Trabajo: Documentación
Esfuerzo estimado: 20
Descripción: Buscar información adicional que permita redactar la memoria además de buscar referencias de otros trabajos o proyectos que puedan servir para corroborar los resultados obtenidos en la fase de experimentación.
Entregas/Salidas: Documento de la memoria del proyecto.
Recursos empleados: Fuentes online de documentación (incluyendo manuales IBM), Microsoft Word

Tabla 2.13 - Tarea búsqueda material de apoyo

4.1.2 Redactar la memoria
Paquete de Trabajo: Documentación
Esfuerzo estimado: 70
Descripción: Redactar la memoria con toda la información referente al desarrollo del proyecto.
Entregas/Salidas: Documento de la memoria del proyecto.
Recursos empleados: Navegador, Microsoft Word

Tabla 2.14 - Tarea Redactar la memoria

4.2 Preparar la defensa
Paquete de Trabajo: Documentación
Esfuerzo estimado: 10
Descripción: Preparar las herramientas necesarias para la presentación y defensa del proyecto.
Entregas/Salidas: Presentación en formato PDF
Recursos empleados: Microsoft Power Point

Tabla 2.15 - Tarea Preparar la defensa

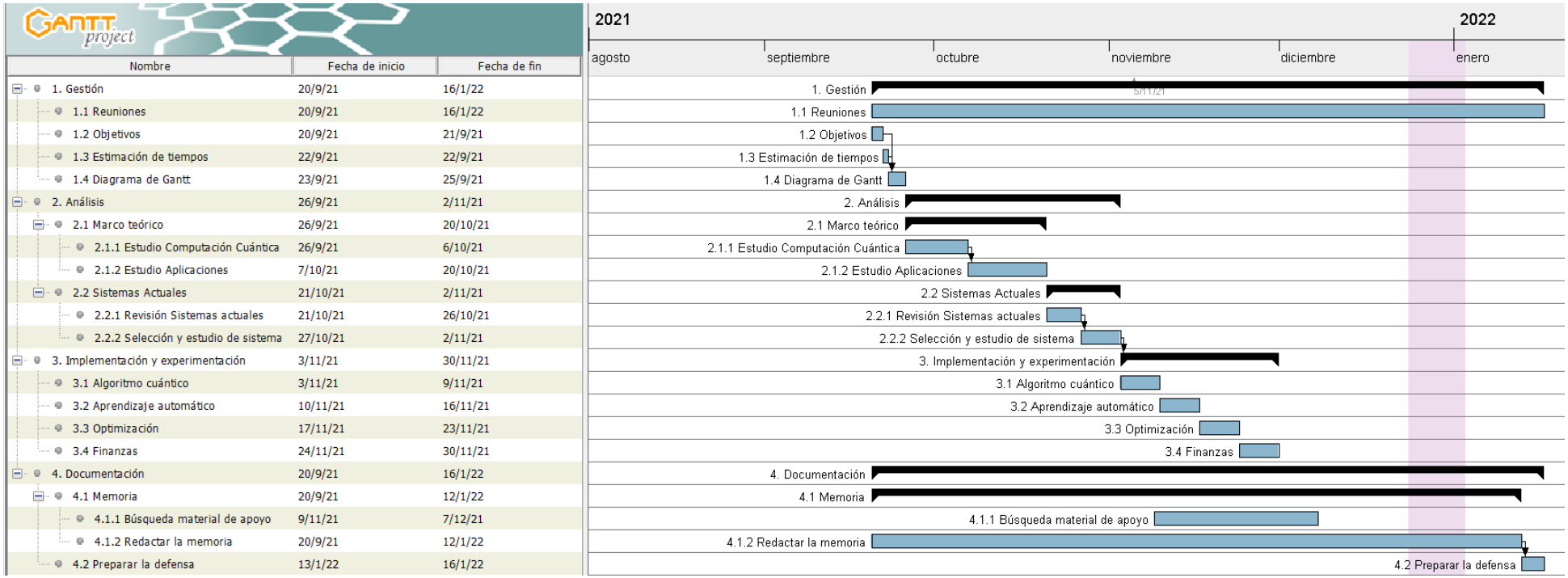
2.3. Planificación temporal

Una vez que se han especificado las tareas a realizar mediante el diagrama EDT, en esta sección se muestra el tiempo planeado de cada uno indicando el esfuerzo necesario para realizar esa tarea en horas y la duración estimada en días (ver Tabla 2.16).

Tareas	Esfuerzo (horas)	Duración (días)
1. Gestión	30	109*
1.1 Reuniones	15	109*
1.2 Objetivos	4	2
1.3 Estimación de tiempos	3	1
1.4 Diagrama de Gantt	8	3
2. Análisis	110	38
2.1 Marco teórico	75	25
2.1.1 Estudio Computación Cuántica	35	11
2.1.2 Estudio Aplicaciones	40	14
2.2 Sistemas Actuales	35	13
2.2.1 Revisión Sistemas actuales	15	6
2.2.2 Selección y estudio de sistema	20	7
3. Implementación y experimentación	80	28
3.1 Algoritmo cuántico	20	7
3.2 Aprendizaje automático	20	7
3.3 Optimización	20	7
3.4 Finanzas	20	7
4. Documentación	100	109*
4.1 Memoria	90	105*
4.1.1 Búsqueda material de apoyo	20	29*
4.1.2 Redactar la memoria	70	26
4.2 Preparar la defensa	10	4
TOTAL	320	109

Tabla 2.16 - Planificación temporal

Las tareas que se van a realizar a lo largo de todo el proyecto o empiezan a la vez que otras tareas pero terminan más tarde han sido marcados con el símbolo *. Por otro lado, teniendo en cuenta todas las tareas, se ha creado el diagrama de Gantt especificando para cada una la fecha de inicio y final esperadas (Ver Figura 2.6).



Nombre	Fecha de inicio	Fecha de fin
1. Gestión	20/9/21	16/1/22
1.1 Reuniones	20/9/21	16/1/22
1.2 Objetivos	20/9/21	21/9/21
1.3 Estimación de tiempos	22/9/21	22/9/21
1.4 Diagrama de Gantt	23/9/21	25/9/21
2. Análisis	26/9/21	2/11/21
2.1 Marco teórico	26/9/21	20/10/21
2.1.1 Estudio Computación Cuántica	26/9/21	6/10/21
2.1.2 Estudio Aplicaciones	7/10/21	20/10/21
2.2 Sistemas Actuales	21/10/21	2/11/21
2.2.1 Revisión Sistemas actuales	21/10/21	26/10/21
2.2.2 Selección y estudio de sistema	27/10/21	2/11/21
3. Implementación y experimentación	3/11/21	30/11/21
3.1 Algoritmo cuántico	3/11/21	9/11/21
3.2 Aprendizaje automático	10/11/21	16/11/21
3.3 Optimización	17/11/21	23/11/21
3.4 Finanzas	24/11/21	30/11/21
4. Documentación	20/9/21	16/1/22
4.1 Memoria	20/9/21	12/1/22
4.1.1 Búsqueda material de apoyo	9/11/21	7/12/21
4.1.2 Redactar la memoria	20/9/21	12/1/22
4.2 Preparar la defensa	13/1/22	16/1/22

Figura 2.6 - Diagrama de Gantt

2.4. Gestión de riesgos

En esta sección se realiza el análisis y la gestión de riesgos del proyecto identificando los imprevistos que pueden llegar a surgir a lo largo del desarrollo del proyecto. El objetivo es poder planificar estrategias que reduzcan el impacto de estos riesgos.

Por lo tanto, en las siguientes tablas (Tabla 2.17 - 2.21) se describe cada riesgo identificado junto con su descripción, como poder prevenirlo, el plan de contingencia, la probabilidad de que ocurra y el impacto que puede causar en el proyecto.

Lesión o enfermedad
Descripción: Enfermedad o lesión en el desarrollador que impida continuar con el desarrollo del trabajo hasta su recuperación.
Prevención: Seguir hábitos de vida saludable y mantener todas las precauciones posibles para evitar enfermarse.
Plan de contingencia: Dependiendo de la gravedad, hablar con el tutor del proyecto para retrasar las tareas y definir nuevas fechas de finalización para cada tarea.
Probabilidad: Baja
Impacto: Medio/Alto

Tabla 2.17 - Riesgo lesión o enfermedad

Problemas del equipo
Descripción: Problemas con el hardware o software utilizado en el desarrollo del proyecto: Ordenador estropeado, perdida de los experimentos realizados, perdida de documentación…
Prevención: Hacer una revisión del estado del ordenador periódicamente para comprobar que no falla nada. Hacer copias de seguridad regularmente en la nube o en otro dispositivo del trabajo realizado.
Plan de contingencia: Sustituir o reparar el ordenador. Recuperar la información de las últimas copias de seguridad realizadas.
Probabilidad: Baja
Impacto: Medio/Alto

Tabla 2.18 - Riesgo problemas del equipo

Cambios en el sistema seleccionado
Descripción: Cambios en las funcionalidades o en la versión del sistema seleccionado para trabajar con las herramientas cuánticas.
Prevención: Mantenerse informado sobre las posibles actualizaciones del sistema o las nuevas funcionalidades incorporadas.
Plan de contingencia: Adaptar las herramientas que se estaban utilizando a la nueva versión.
Probabilidad: Baja
Impacto: Medio

Tabla 2.19 - Riesgo cambios en el sistema seleccionado

Falta de experiencia
Descripción: Falta de experiencia en el uso de las herramientas en áreas como el aprendizaje automático, la optimización o finanzas. La falta de experiencia en tecnologías o la teoría referente a la computación cuántica es algo con los que se parte desde el principio y se irá mitigando a lo largo del proyecto.
Prevención: Dedicar más tiempo a buscar información o realizar más pruebas.
Plan de contingencia: Aplicar los conocimientos adquiridos tras la búsqueda adicional de información.
Probabilidad: Alto
Impacto: Medio

Tabla 2.20 - Riesgo falta de experiencia

Problemas de conexión de internet
Descripción: Corte o caída temporal de la conexión a internet, imposibilitando el acceso a recursos online para la búsqueda de información o el acceso al sistema con el que se esté trabajando.
Prevención: Utilizar líneas de conexión estables que puedan mantener una buena velocidad de conexión.
Plan de contingencia: Utilizar otro punto de acceso a internet o pasar a hacer otra tarea que no requiera conexión a internet hasta que se pueda recuperar.
Probabilidad: Medio
Impacto: Muy bajo

Tabla 2.21 - Riesgo problemas de conexión de internet

2.5. Evaluación económica

En esta sección se realiza la evaluación económica con los costes que supone el proyecto. Entre ellos, se estima los costes de los trabajadores, hardware, software y costes indirectos como la electricidad o la conexión de internet.

Al ser un trabajo de investigación para el TFG no se espera un beneficio económico, sin embargo, realizar esta evaluación puede servir para un trabajo futuro en caso de que se quiera adaptar o expandir y tener una base sobre lo que costaría.

2.5.1. Mano de obra

En primer lugar, se encuentran los costes por mano de obra o los trabajadores del proyecto. Un ingeniero informático que ha acabado recientemente un grado y con menos de 3 años de experiencia laboral tiene un salario medio alrededor de 20.450€ brutos al año [5]. A este valor hay que aplicarle las reducciones económicas para obtener el salario neto, tales como el IRPF y las cuotas de seguridad social.

Con una retención por IRPF del 11.92% se retiene 2437.64€ al año, y con una cotización del 6.35% a la seguridad social se debe aportar 1298.575 € al año [6]. Por lo tanto, restando estos valores se obtiene el salario neto al año de un ingeniero informático recién graduado: 20450€ (Salario bruto mensual) – 2437.64€ (IRPF) – 1298.575€ (Seguridad Social) = 16714.425 € (Neto), lo que equivale a 1392.87 € netos al mes.

Suponiendo que en una jornada normal se trabaja 40 horas a la semana y que un mes tiene 4 semanas, en un mes se harían 160 horas de trabajo. Por lo tanto, dividiendo el sueldo al mes con este valor se obtiene el sueldo por hora: 1392.87€ / 160 = 8.71€/h.

A este valor se le debe añadir los costes que suponen a una empresa el pago de la seguridad social, siendo estos los siguientes: contingencias comunes (23,6%), cotización por formación (0.6%), cotización por desempleo (5.5 % en contratos indefinidos), cotización al fondo de garantía salarial (0,2%) y cotización por accidentes de trabajo y enfermedades (1%) [7]. Por lo tanto, se debe añadir un coste adicional del 30.9% al coste por hora del trabajador, obteniendo un valor de 11.40 €/h.

Adicionalmente, se debe tener en cuenta los costes supuestos por el tutor del proyecto, por lo que se ha obtenido el sueldo medio de profesores de universidad en España, con un salario bruto medio de 43400€ [8]. Al igual que con el sueldo de ingeniero, después de aplicar las retenciones necesarias se obtiene un sueldo neto de 2641.6 €/mes, lo que equivale a 16,51€/h. Añadiendo los costes de la seguridad social que le suponen a una empresa, se obtiene el coste final de 21.62€/h.

Por lo tanto, teniendo en cuenta que el esfuerzo estimado del proyecto son 320 horas y el tutor del proyecto tiene una dedicación aproximada de 30 horas entre las reuniones, la revisión del trabajo y la resolución de dudas o preguntas, se obtiene el coste total de los trabajadores: 11.40€/h * 320h + 21.62€/h * 30 = 4296.6 €

2.5.2. Recursos utilizados

Una vez obtenido el coste del personal, se debe tener en cuenta los costes de los recursos utilizados, lo cual incluye el coste del hardware utilizado y los costes relacionados con el software o las herramientas.

En cuanto al **hardware**, se ha utilizado un ordenador valorado en 700€ y una vida útil de 4 años, por lo que la amortización al año es de 175€. Teniendo en cuenta este dato y sabiendo que se ha utilizado a lo largo de todo el proyecto, se obtiene el coste total del hardware: 175€ * (116/365) = 55.61€

En cuanto al **software** y las herramientas utilizadas no ha habido ningún coste, ya que se han utilizado programas gratuitos o los que ya incluía el propio ordenador. Además, para el estudio y selección de las plataformas de computación cuántica se ha accedido únicamente a los servicios gratuitos.

2.5.3. Costes indirectos

Por último, hay que tener en cuenta los **costes indirectos**, es decir, los recursos que se utilizan durante el proyecto pero que no afectan directamente al resultado final, como la electricidad o la conexión a internet. Para ello, se puede hacer un cálculo aproximado aplicando un 2% a los demás costes, es decir: (4296.6 € + 55.61€) * 2% = 87.04€

2.5.4. Costes totales

En definitiva, todos los costes del proyecto han sido los siguientes:

- Coste total del personal: 4296.6€
- Costes de hardware: 55.61€
- Costes de software: 0€
- Costes indirectos: 87.04€

Por lo tanto, teniendo en cuenta los costes de personal, hardware, software y costes indirectos, se obtiene un **coste total de 4439.25€.**

3. TEORIA DE LA COMPUTACIÓN CUÁNTICA

En esta sección se exponen los fundamentos matemáticos y de computación cuántica básicos necesarias para el desarrollo del trabajo. Entre otras cosas, se analiza la diferencia de un bit clásico frente a su análogo cuántico qubit y como representarlos, el uso de puertas cuánticas y el análisis de algunos fenómenos cuánticos como la superposición o el entrelazamiento. Como referencia principal, se han utilizado las guías de IBM Q y Qiskit para el aprendizaje [9] [10].

3.1. Bit vs Qubit

En la computación clásica, disponemos de bits como la unidad de información más básica. Un bit es un número binario que puede estar en uno de sus 2 valores posibles, 0 o 1. También se puede entender el concepto de bit como verdadero o falso, encendido o apagado o cualquier otra similitud que tome 2 valores.

En la computación cuántica, en lugar de bits se tienen los qubits o bits cuánticos, los cuales, al igual que su análogo clásico, también tienen 2 estados simples, 0 o 1. La diferencia es, que un qubit también puede estar en una proporción de ambos estados a la vez, con una probabilidad concreta de ser 0 y otra probabilidad de ser 1. Este fenómeno de poder tener ambos estados a la vez es lo que se denomina **superposición**.

Matemáticamente, el qubit se puede representar como un vector de módulo unidad en un espacio vectorial complejo bidimensional [11]. En computación cuántica los estados de un qubit se pueden denotar utilizando la notación de Dirac, teniendo como los dos estados básicos de un qubit el $|0\rangle$ (ket cero) y el $|1\rangle$ (ket uno), los cuales corresponden al 0 y 1 del bit clásico. Además, como se ha mencionado que un qubit se puede encontrar en una superposición de ambos estados, se puede definir de la siguiente forma:

$$|\Psi\rangle = \alpha\,|0\rangle + \beta|1\rangle$$

En esta fórmula, podemos observar que el estado $|\Psi\rangle$ de un qubit cualquiera está representado por las amplitudes α y β. Utilizando el módulo al cuadrado de estos valores($|\alpha|^2$ y $|\beta|^2$), se obtiene la probabilidad del estado $|0\rangle$ y el estado $|1\rangle$ respectivamente, y la suma de ambos debe de ser 1.

Teniendo en cuenta α y β , se puede escribir el estado de un qubit con forma de vector columna $\begin{pmatrix}\alpha\\ \beta\end{pmatrix}$. De esta forma, $|0\rangle$ se puede representar como $\begin{pmatrix}1\\ 0\end{pmatrix}$ y que $|1\rangle$ se puede representar como $\begin{pmatrix}0\\ 1\end{pmatrix}$, ya que en el primer caso indicamos que se tiene un 100% de probabilidad de ser el estado $|0\rangle$ y en el segundo caso un 100% de probabilidad de ser el estado $|1\rangle$.

Cuando se mide un qubit en un estado cualquiera, se dice que este colapsa hacia uno de los estados posibles, es decir, toma uno de los valores posibles dependiendo de la probabilidad de sus amplitudes. Por ejemplo, al medir el qubit en el estado $|\Psi\rangle$ podría colapsar hacia el estado $|0\rangle$ o hacia el estado $|1\rangle$**.** Una vez ha colapsado hacia uno de esos estados, si se vuelve a medir se obtendrá siempre el mismo estado otra vez, por lo que observar el estado o medir un qubit cambia el estado para siempre.

Además, α y β son números complejos, por lo que se puede representar el estado de un qubit como una esfera, más concretamente con la esfera de Bloch (ver Figura 3.1).

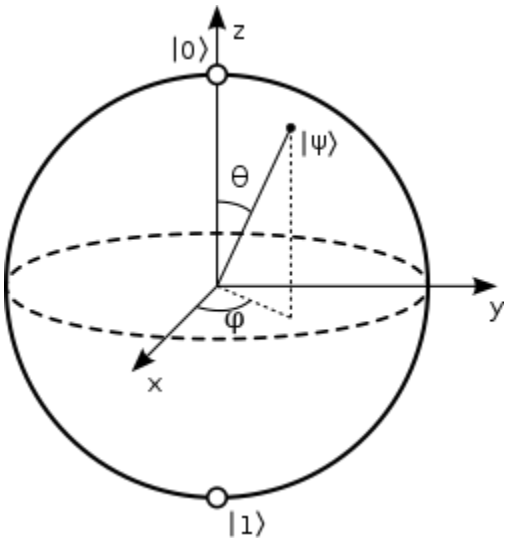


Figura 3.1 - Representación de un qubit mediante la esfera de Bloch [11]

Esta esfera está construida de forma que en polo norte queda el estado $|0\rangle$ y en el polo sur el estado $|1\rangle$. El resto de posibles estados quedarían representados a lo largo de la superficie de la esfera.

Adicionalmente, un estado será más probable que colapse hacia el estado $|0\rangle$ cuanto esté más al norte, y más probable que colapse hacia el estado $|1\rangle$ cuando esté más al sur. Aunque se menciona más adelante, cuando el qubit se encuentra en una superposición equiprobable (tiene la misma probabilidad de colapsar al estado $|0\rangle$ o $|1\rangle$) este se encuentra en el estado $|+\rangle$ (ver Figura 3.2).

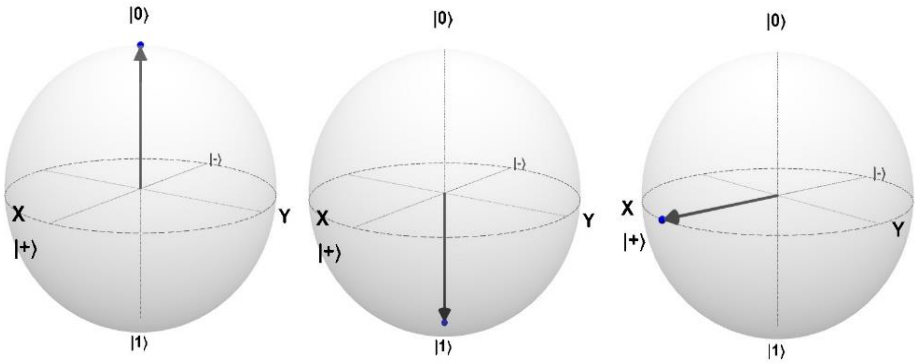


Figura 3.2 - Esfera de Bloch en diferentes estados del qubit [77]

En el caso de trabajar con 2 qubits, el estado se modeliza con un vector de números complejos de 4 elementos:

$$|\Psi_1\Psi_2\rangle = \alpha\,|00\rangle + \beta|01\rangle + \gamma|10\rangle + \delta|11\rangle$$

$$|\Psi_1\Psi_2\rangle = \begin{pmatrix} \alpha \\ \beta \\ \gamma \\ \delta \end{pmatrix}$$

Por lo tanto, con 2 qubits se tienen las siguientes probabilidades, con las que se mantiene que la suma de los módulos al cuadrado es uno.

$$p(|00\rangle) = |\alpha|^2$$
$$p(|01\rangle) = |\beta|^2$$
$$p(|10\rangle) = |\gamma|^2$$
$$p(|11\rangle) = |\delta|^2$$

$$|\alpha|^2 + |\beta|^2 + |\gamma|^2 + |\delta|^2 = 1$$

Para llegar al estado conjunto de 2 qubits, se utiliza el producto de Kronecker, un producto combinatorio entre matrices [12]. Lo que se consigue es multiplicar cada elemento de la primera matriz con los elementos de la segunda matriz:

$$|\Psi_1\Psi_2\rangle = \begin{pmatrix}\alpha_1\\ \beta_1\end{pmatrix} \otimes \begin{pmatrix}\alpha_2\\ \beta_2\end{pmatrix} = \begin{pmatrix}\alpha_1 \cdot \begin{pmatrix}\alpha_2\\ \beta_2\end{pmatrix}\\ \beta_1 \cdot \begin{pmatrix}\alpha_2\\ \beta_2\end{pmatrix}\end{pmatrix} = \begin{pmatrix}\alpha_1\alpha_2\\ \alpha_1\beta_2\\ \beta_1\alpha_2\\ \beta_1\beta_2\end{pmatrix} = \begin{pmatrix}\alpha\\ \beta\\ \gamma\\ \delta\end{pmatrix}$$

En el caso de tener n qubits, se pueden representar 2^n estados posibles, por lo que ya podemos ver una ventaja respecto a los bits clásicos. Mientras que un registro de n bits solamente puede estar en uno de los 2^n estados posibles, un registro de qubits de la misma longitud puede teóricamente estar en cualquier superposición de todos ellos. Sin embargo, al medir el estado final solamente se puede obtener uno de sus valores posibles.

Por otro lado, al tener todos los estados posibles en un estado de superposición, es posible aplicar una función con sus respectivas variables que transforme el estado obteniendo así en un único paso todos los valores de la función.

Esta función en computación cuántica es comúnmente utilizada y se le llama **oráculo**. Estos oráculos son representados por circuitos cuánticos que realizan una función dada una entrada y transforman el estado de los qubits para producir la salida correspondiente.

En resumen, la mayor diferencia entre los bits clásicos y los qubits es que mientras los bits solamente pueden estar en uno de sus valores posibles, los qubits pueden estar en una superposición de todos esos estados a la vez (ver Figura 3.3).

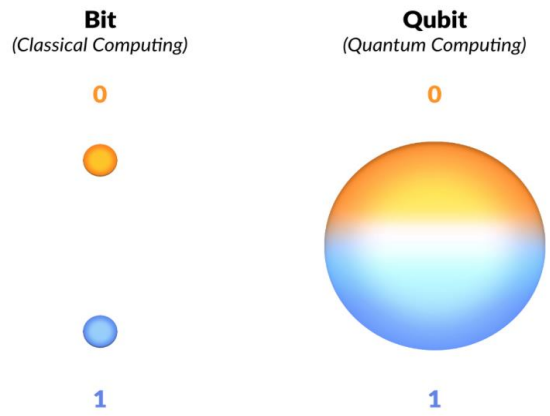


Figura 3.3 - Diferencia visual entre bit clásico y qubit [78]

3.2. Puertas cuánticas

En la computación clásica se puede entender una puerta lógica como una función booleana que toma un conjunto de bits y los procesa para obtener una salida que será o un 0 o un 1. Por ejemplo, la puerta lógica NOT (ver Figura 3.4) transforma un bit en su complementario:

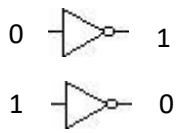


Figura 3.4 - Puerta lógica NOT

Entrada	Salida
0	1
1	0

En cambio, en la computación cuántica, se hace uso de las puertas lógicas cuánticas o, dicho de otro modo, puertas cuánticas. Estas puertas son conceptualmente análogas a las clásicas, ya que reciben un registro de qubits en un determinado estado y aplican una transformación sobre ese estado para transformarlo.

Esto es una diferencia respecto a las puertas lógicas clásicas, ya que mientras en estas el bit de origen queda intacto al aplicar las puertas, con una puerta cuántica el qubit original se modifica cambiando su estado cuando se aplica dicha puerta.

Estas puertas se modelan como una matriz de números complejos, y al utilizarlos para transformar el estado del qubit no colapsan los estados y no alteran la suma de los cuadrados de los módulos de los elementos del vector estado, es decir, la suma de probabilidad sigue siendo uno.

El conjunto de estas matrices se denomina matrices unitarias, los cuales tienen la propiedad de que al multiplicarlas por la conjunta de su transpuesta nos da la identidad. Además, las puertas cuánticas son reversibles, por lo que, si la misma puerta se aplica dos veces seguidas, el estado cuántico que se obtiene es el inicial.

Una de las puertas cuánticas más simples es la **puerta X**, la cual se puede entender como el análogo a la puerta lógica NOT en la computación clásica, ya que transforma el $|0\rangle$ en el $|1\rangle$ y $|1\rangle$ en el $|0\rangle$, aunque lo que realmente hace esta puerta es intercambiar las probabilidades de lectura de uno y cero.

La matriz que lo representa es la siguiente:

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

Para entender mejor cómo funciona la transformación de un qubit utilizando las puertas cuánticas, se puede observar un ejemplo en el que se aplica la puerta X a un estado cualquiera:

$$|\varphi\rangle = \begin{pmatrix} \alpha \\ \beta \end{pmatrix}$$

$$|\varphi'\rangle = \mathbf{X}|\varphi\rangle = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} \alpha \\ \beta \end{pmatrix} = \begin{pmatrix} 0\alpha + 1\beta \\ 1\alpha + 0\beta \end{pmatrix} = \begin{pmatrix} \beta \\ \alpha \end{pmatrix}$$

Como se puede ver, la transformación se realiza multiplicando la matriz por el vector y en la matriz final quedan intercambiadas los parámetros α y β, lo cual sería el nuevo estado del qubit.

Otra puerta cuántica y una de las más importantes en la computación cuántica, es la **puerta Hadamard (H)**. Esta puerta pone en un estado de superposición al qubit y su matriz es la siguiente:

$$H = \frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}$$

Al igual que con la puerta X, la transformación del qubit se realiza multiplicando la matriz de la puerta por su estado.

$$|\varphi\rangle = \begin{pmatrix}\alpha\\ \beta\end{pmatrix}$$

$$|\varphi'\rangle = \mathbf{H}|\varphi\rangle = \frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}\begin{pmatrix}\alpha\\ \beta\end{pmatrix} = \frac{1}{\sqrt{2}}\begin{pmatrix}\alpha + \beta\\ \alpha - \beta\end{pmatrix}$$

Para entender mejor como se consigue y la superposición que crea, se puede observar lo que ocurre cuando se aplica la puerta H en los estados $|0\rangle$ y $|1\rangle$:

$$|0\rangle = \begin{pmatrix}1\\ 0\end{pmatrix}$$

$$\mathbf{H}|0\rangle = \frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}\begin{pmatrix}1\\ 0\end{pmatrix} = \frac{1}{\sqrt{2}}\begin{pmatrix}1\\ 1\end{pmatrix}$$

$$p(|0\rangle) = \left(\frac{1}{\sqrt{2}}\right)^2 = 0.5$$

$$p(|1\rangle) = \left(\frac{1}{\sqrt{2}}\right)^2 = 0.5$$

Al medir el estado se obtiene un 50% de probabilidad para el estado $|0\rangle$ y un 50% de probabilidad para el estado $|1\rangle$, lo cual es una superposición equiprobable. Como se ha visto anteriormente en la Figura 3.2, este estado se denota $|+\rangle$ (ket positivo) y el nuevo estado se puede representar como $\frac{|0\rangle + |1\rangle}{\sqrt{2}}$.

En el caso de aplicar la puerta Hadamard al estado inicial $|1\rangle$ las probabilidades finales siguen siendo las mismas:

$$|0\rangle = \begin{pmatrix}0\\ 1\end{pmatrix}$$

$$\mathrm{H}|0\rangle = \frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}\begin{pmatrix}0\\ 1\end{pmatrix} = \frac{1}{\sqrt{2}}\begin{pmatrix}1\\ -1\end{pmatrix}$$

$$p(|0\rangle) = \left(\frac{1}{\sqrt{2}}\right)^2 = 0.5$$

$$p(|1\rangle) = \left(-\frac{1}{\sqrt{2}}\right)^2 = 0.5$$

Las probabilidades siguen siendo 0.5 para cada estado, ya que con el módulo al cuadrado no afecta que tengamos un valor negativo en la probabilidad de obtener el estado $|1\rangle$. Aunque las probabilidades sean las mismas y se encuentre en un estado de superposición equiprobable, al tener la amplitud del estado $|1\rangle$ en negativo se dice que se encuentra en el estado $|-\rangle$ (ket negativo) y se puede representar como $\frac{|0\rangle - |1\rangle}{\sqrt{2}}$. Como se va a ver más adelante en el algoritmo de Grover, esto puede ser utilizado para marcar un estado concreto con una amplitud negativa.

Otra puerta cuántica básica comúnmente utilizada es la puerta **Pauli Z.** Esta puerta transforma el $|+\rangle$ en el $|-\rangle$ y el $|-\rangle$ en el $|+\rangle$. También se puede entender como un mapeo del estado $|1\rangle$ al -$|1\rangle$ dejando el $|0\rangle$ inalterado. La matriz que representa esta puerta es la siguiente:

$$Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

También existen otras puertas cuánticas que permiten cambiar la fase de un qubit, es decir, pueden aplicar una rotación a lo largo de la esfera de Bloch. Como ejemplo, la **puerta P**, recibe un parámetro que será utilizado para aplicar la rotación. Su matriz se puede ver a continuación:

$$P(\Phi) = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\Phi} \end{pmatrix}$$

Una de las puertas cuánticas que derivan de esta última es **la puerta S**, el cual hace una rotación de $\frac{\pi}{2}$ sobre la esfera de Bloch, siendo su respectiva matriz la siguiente:

$$S = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\frac{\pi}{2}} \end{pmatrix}$$

En la Figura 3.5 se puede ver el uso de esta puerta al aplicarse a un qubit en estado de superposición:

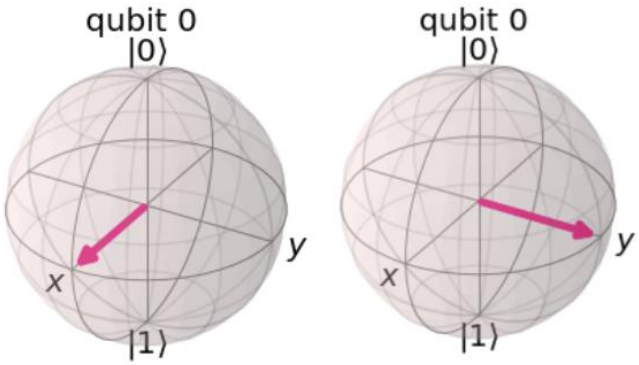


Figura 3.5 - Rotación de un cuarto en la esfera de Bloch al aplicar la puerta S

Uno de los usos de estas últimas puertas es una de las formas en las que se pueden codificar variables clásicas en un entorno cuántico, poniendo una rotación con un ángulo distinto en los qubits. Existen más puertas cuánticas que permiten cambiar el estado de un qubit, las cuales se pueden ver en la documentación de Qiskit junto con más información o ejemplos sobre los ya mencionados [13].

Las puertas que se han mencionado hasta ahora eran puertas aplicadas a un único qubit, pero existen otras puertas que se aplican a más. Una de ellas, es la **puerta CNOT** o Controlled NOT (Ver Figura 3.6), que se aplica sobre 2 qubits. En el caso de aplicarlo a un estado cualquiera $|\varphi_1\varphi_2\rangle$**,** el primer qubit ($\varphi_1$) se denomina el control y el segundo qubit (φ_2) se denomina objetivo.

El efecto de esta puerta es que cuando el control se encuentra en el estado $|0\rangle$, no se hace ningún cambio en el qubit objetivo. Sin embargo, si el qubit control se encuentra en el estado $|1\rangle$, se aplica la puerta X en el qubit objetivo.

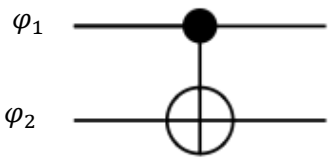


Figura 3.6 - Puerta CNOT

$\|\varphi_1\varphi_2\rangle$		CNOT$\|\varphi_1\varphi_2\rangle$	
Control	**Objetivo**	**Control**	**Objetivo**
$\|0\rangle$	$\|0\rangle$	$\|0\rangle$	$\|0\rangle$
$\|0\rangle$	$\|1\rangle$	$\|0\rangle$	$\|1\rangle$
$\|1\rangle$	$\|0\rangle$	$\|1\rangle$	$\|1\rangle$
$\|1\rangle$	$\|1\rangle$	$\|1\rangle$	$\|0\rangle$

La matriz que representa esta puerta es la siguiente:

$$CNOT = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

Al igual que las demás puertas, la transformación del qubit se realiza multiplicando la matriz de la puerta por su estado. A continuación, se puede ver un ejemplo al aplicarlo a un estado cualquiera, en el que se cambian las probabilidades de los estados $|10\rangle$ y $|11\rangle$.

$$\text{CNOT}|\varphi_1\varphi_2\rangle = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} \alpha_1\alpha_2 \\ \alpha_1\beta_2 \\ \beta_1\alpha_2 \\ \beta_1\beta_2 \end{pmatrix} = \begin{pmatrix} \alpha_1\alpha_2 \\ \alpha_1\beta_1 \\ \beta_1\beta_2 \\ \beta_1\alpha_2 \end{pmatrix}$$

De la misma forma que esta puerta se aplica a 2 qubits, existe la puerta Toffoli, la cual es aplicada a 3 puertas. En este caso, cuando los 2 primeros qubits se encuentran en el estado $|11\rangle$ se aplica la puerta X al tercer qubit.

Una vez que se han analizado algunas puertas cuánticas, se puede definir un circuito cuántico. Los circuitos cuánticos son concatenaciones de las puertas cuánticas que producen cambios de estado en los qubits de una forma ordenada.

En la notación gráfica, se expresan mediante cables horizontales, siendo un cable por cada qubit. Encima de estos cables y en un orden determinado se localizan las puertas cuánticas que irán transformando el estado del qubit. Como ejemplo, se puede ver un circuito cuántico en la Figura 3.7 que utiliza una puerta Hadamard y una puerta CNOT, siendo el control el primer qubit.

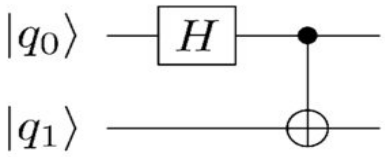


Figura 3.7 - Circuito cuántico

Siguiendo el circuito del ejemplo, los estados se transforman de la siguiente forma:

- $|\varphi_1\rangle = |00\rangle$
- $|\varphi_2\rangle = \text{H}|\varphi_1\rangle = (\frac{\mathbf{1}}{\sqrt{\mathbf{2}}}(|0\rangle + |1\rangle)|0\rangle = \frac{\mathbf{1}}{\sqrt{\mathbf{2}}}(|00\rangle + |10\rangle)$
- $|\varphi_3\rangle = \text{CNOT}|\varphi_2\rangle = \frac{\mathbf{1}}{\sqrt{\mathbf{2}}}(|00\rangle + |11\rangle)$

Por lo tanto, el estado final que se obtiene es $\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$ y si se mide este circuito concreto, se obtiene un 50% de probabilidad para el estado $|00\rangle$ y un 50% de probabilidad para el estado $|11\rangle$.

Sin embargo, tal y como se va a ver a continuación, este ha sido un circuito concreto que obtiene un estado especial. Como se ha explicado anteriormente, el estado conjunto de 2 qubits se obtiene con el producto de los 2 estados.

$$|\varphi_1\varphi_2\rangle = \begin{pmatrix}\alpha_1\\ \beta_1\end{pmatrix} \otimes \begin{pmatrix}\alpha_2\\ \beta_2\end{pmatrix} = \begin{pmatrix}\alpha_1\alpha_2\\ \alpha_1\beta_2\\ \beta_1\alpha_2\\ \beta_1\beta_2\end{pmatrix}$$

Por lo tanto, se puede igualar esta expresión con el estado que se ha obtenido al final del circuito:

$$\begin{pmatrix}\alpha_1\alpha_2\\ \alpha_1\beta_2\\ \beta_1\alpha_2\\ \beta_1\beta_2\end{pmatrix} = \frac{1}{\sqrt{2}}\begin{pmatrix}1\\ 0\\ 0\\ 1\end{pmatrix}$$

Sin embargo, al intentar igualar ambas expresiones se llega a una contradicción. Si $\alpha_1\beta_2 = 0$, α_1 debe de ser 0 y no se puede cumplir la condición de $\alpha_1\alpha_2 = \frac{1}{\sqrt{2}}$. Por otro lado, si $\beta_1\beta_2 = 0$, β_1 debe ser 0 por lo que tampoco se puede cumplir la condición de $\beta_1\beta_2 = \frac{1}{\sqrt{2}}$.

En definitiva, este estado que se ha creado en el circuito, no se puede conseguir con la simple combinación de dos estados individuales, es decir, no se puede descomponer como el producto de los estados. Este estado generado se llama estado de Bell, el cual representa un ejemplo del **entrelazamiento cuántico**.

A causa de este fenómeno, aunque los qubits se encuentren en superposición, si se mide uno de los qubits nos da la información sobre el otro qubit, colapsando el estado de superposición. Por ejemplo, si al medir el primer qubit se obtiene el estado $|1\rangle$**,** el estado colapsa hacia el $|11\rangle$**.** Por lo tanto, si dos qubits se encuentran entrelazados, aunque se separen físicamente, medir uno de ellos afecta directamente en el otro.

3.3. Decoherencia cuática

Hasta ahora se han visto algunas de las bases de la computación cuántica y las ventajas que puede ofrecer como la superposición, permitiendo obtener muchos estados a la vez con unos pocos qubits. Sin embargo, las computadoras cuánticas actuales sufren de varios problemas y en ocasiones no se consiguen los resultados que se esperan.

Para un estado de un qubit cualquiera, el tiempo en el que permanezca en ese estado sin alterarse es el tiempo de coherencia. En los ordenadores cuánticos actuales, este estado puede verse alterado por el ruido proveniente del entorno en el que se encuentra ese qubit, como fluctuaciones en la temperatura, vibraciones, ondas electromagnéticas o impurezas del propio material del qubit [14].

A esta alteración del estado, se le llama decoherencia. Es decir, la decoherencia cuántica ocurre cuando los qubits pierden información a causa de los errores ocasionados en el sistema en un periodo de tiempo.

Además, a medida que las aplicaciones aumentan en tamaño y en complejidad, los sistemas se vuelven más propensos a sufrir este problema y cuanto más tiempo pasa desde que se inicia un algoritmo cuántico, los qubits sufren más estas alteraciones, por lo que los resultados finales pueden no ser los esperados.

Como ejemplo, si se tiene que el estado $|0\rangle$ es el estado base y ha sido transformado al estado $|1\rangle$ utilizando la puerta X, es posible que con el paso del tiempo vuelva al estado $|0\rangle$, por lo que se altera el estado en el que estábamos. Para otros estados que estén en superposición como el $|+\rangle$**,** en lugar de mantenerse las mismas probabilidades para los 2 estados posibles, estos podrían verse alterados y cambiar las amplitudes a favor o en contra de cualquiera de ellos.

Para evitar que los qubits cambien de estado cuántico de forma espontánea, se deben aislar los qubits de la mejor manera posible, entre ellos estableciendo una temperatura lo más cercana al cero absoluto. En el caso de IBM Q, los ordenadores cuánticos trabajan con alrededor de 15 milikelvin [15], lo que son aproximadamente -273 grados Celsius.

Por otro lado, para reducir el tiempo de ejecución en los sistemas cuánticos y reducir la decoherencia, se pueden utilizar algoritmos híbridos cuánticos-clásicos, en los que se ejecuta únicamente las partes más críticas en cuanto al rendimiento de un programa en los sistemas cuánticos y la mayoría del programa se ejecuta en un ordenador clásico.

Además, también existen métodos o aplicaciones que permiten corregir estos errores. Sin embargo, son necesarios utilizar varios qubits para ello, por lo que se reduce la cantidad de qubits utilizados para el cálculo real y se reduce el tamaño de la tarea informática a una pequeña fracción de lo que podría ejecutarse en un hardware sin defectos.

4. SISTEMAS ACTUALES

Dia a día la computación cuántica avanza más y algunas empresas ponen a disposición de los usuarios herramientas para poder utilizar distintos servicios relacionados con esta tecnología. Algunas de estas empresas, disponen de lenguajes propios de programación y permiten ejecutar programas en máquinas cuánticas reales, mientras que otros solamente disponen de simuladores.

El objetivo de esta sección es hacer un breve análisis de algunos de estos sistemas que ofrecen las empresas y una vez observados algunos de los más famosos, seleccionar uno de ellos para hacer un análisis más exhaustivo y poder trabajar con él.

Una de estas empresas es IBM, que mediante su servicio **IBM Quantum** (IBM Q) ofrece una plataforma online de acceso público y privado para la computación cuántica [16]. Esta plataforma incluye acceso a distintos sistemas cuánticos reales además de simuladores. Todos los usuarios que se creen una cuenta en IBM Quantum tienen acceso a estos servicios de manera gratuita, pero en el caso de los sistemas cuánticos reales solamente disponen de 6 sistemas de máximo 5 qubits.

Sin embargo, IBM Q dispone de *IBM Quantum Network* [17], una comunidad en la que se encuentran muchas compañías, academias institucionales, laboratorios y empresas recién establecidas y que dispone de sistemas cuánticos con mayor cantidad de qubits además de otras ventajas. Para unirse a esta comunidad es necesario hacer una solicitud indicando a que industria se pertenece.

Por otro lado, para trabajar con las computadoras cuánticas, IBM Q dispone de un kit de desarrollo propio llamado Qiskit. Con Qiskit [18], el cual utiliza el lenguaje de programación Python, es posible la creación y manipulación de los programas cuánticos para su posterior ejecución en los dispositivos cuánticos de IBM Q o en simuladores. Además, IBM Q ofrece gran cantidad de documentación en forma de tutoriales o programas que se pueden ejecutar como cuadernos de Jupyter utilizando Qiskit.

Otra de las empresas que ofrecen este tipo de herramientas es Amazon con su servicio **Amazon Braket**, el cual es parte de Amazon Web Services (AWS), una colección de servicios de computación en la nube pública [19]. Su nombre viene dado por la notación bra-ket (o notación de Dirac) mencionada anteriormente en la sección de teoría cuántica.

AWS ofrece una capa gratuita y otra de pago, sin embargo, la capa gratuita está limitada por tiempo de uso, y una vez excedido este límite, se sigue la dinámica de pago sobre la marcha, en el que se paga la cantidad correspondiente al uso de cada sistema [20]. Amazon Braket entra dentro de ese sistema, ofreciendo una hora gratuita de simulación de circuitos cuánticos al mes y durante los primeros 12 meses.

Al igual que IBM Q, Amazon Braket ofrece simuladores y sistemas reales para la ejecución de programas cuánticos, siendo los sistemas reales los de Riggetti, IonQ y D-Wave. Se puede trabajar con estas herramientas utilizando cuadernos Jupyter después de instalar el kit de desarrollo de Amazon Braket.

Microsoft también ofrece herramientas de computación cuántica con su servicio Azure Quantum [21], el cual es parte de Azure, un conjunto de servicios en la nube de Microsoft. Para trabajar con estas herramientas, Microsoft dispone de un kit de desarrollo de código abierto que

incluye un lenguaje de programación cuántico llamado Q# y al igual que con Amazon Braket y Qiskit se puede trabajar utilizando los cuadernos Jupyter o con las extensiones de Visual Studio y Visual Studio Code.

Para ejecutar los programas cuánticos, Azure Quantum dispone tanto de simuladores como de proveedores de computación cuántica como IonQ y HoneyWell [22] y sigue la dinámica de pago según el uso.

También existen biblioteca de código abierto como **PennyLane** [23], desarrollado por Xanadu Quantum Technologies, que permite la programación diferenciable de computadoras cuánticas. Tiene compatibilidad con modelos híbridos cuánticos que usan una parte cuántica y otra parte clásica, además de compatibilidad con las bibliotecas de aprendizaje automático existentes. Además, tiene la posibilidad de ejecutar un modelo de un circuito cuántico en distintos dispositivos después de instalar los complementos para ejecutarlos, como en Amazon Braket, Qiskit o Microsoft QDK.

Existen más empresas y kit de desarrollos que se pueden utilizar para la creación y manipulación de los programas cuánticos como Forest (desarrollado por Riggetti); Ocean (desarrollado por D-wave) o incluso Circq que es desarrollado por Google [24].

Como se ve, son muchos las empresas que tienen este tipo de herramientas, y cada vez irán surgiendo más con el objetivo de atraer más personas y expandir el conocimiento que se tiene sobre la computación cuántica.

Entre estos sistemas, se ha decidido trabajar con IBM Q y Qiskit para un análisis más profundo de los sistemas. Una de las razones de esta selección ha sido la gran variedad de documentación que se ha encontrado en el estudio inicial de la computación cuántica, además de tener claramente separadas varias secciones que permiten probar aplicaciones en distintos ámbitos como el aprendizaje automático o la optimización, que es el objetivo del proyecto.

Además, mientras que en otros sistemas que se han observado es necesario pagar según el uso que se les da, en este caso todas las funcionalidades son gratis para cualquier cuenta. Adicionalmente, se disponía de una cuenta educacional que incluye ciertas ventajas que se mencionan más adelante.

A continuación, se hace un análisis más profundo de algunas de las características que contienen IBM Q y Qiskit.

4.1. IBM Q

Como ya se ha mencionado, IBM Q (IBM Quantum), es el servicio ofrecido por IBM para la computación cuántica. En esta sección se analizan algunos de sus componentes como los proveedores, el funcionamiento de enviar trabajos a sus sistemas o las herramientas Composer y Lab.

4.1.1. Proveedores

El acceso a los distintos servicios ofrecidos por IBM Quantum es controlado mediante los proveedores asignados a cada persona [25]. Un proveedor es una estructura jerárquica organizada por el centro, grupo y proyecto. El centro es el nivel más alto y puede contener varios grupos, mientras que los grupos pueden estar compuestos por distintos proyectos. Esta combinación de centro/grupo/proyecto es denominado el proveedor y por ejemplo permite

gestionar a los administradores de los proveedores los permisos de uso en sistemas concretos a los usuarios de un grupo o proyecto.

Cualquier usuario que se cree una cuenta en IBM Q es añadido automáticamente al proveedor público ibm-q/open/main, el cual dispone de 6 sistemas cuánticos reales de 5 qubits (ibmq_manila, ibmq_bogota, ibmq_santiago, ibmq_quito, ibmq_belem, ibmq_lima) y ibmq_armonk de 1 qubit, además de simuladores con más cantidad de qubits que también se pueden utilizar.

Además de los sistemas que se pueden utilizar, cada proveedor tiene un límite distinto de circuitos que se pueden enviar a un sistema en un mismo tiempo o la cantidad máxima de repeticiones que se pueden ejecutar de un circuito en un sistema.

Para este trabajo, se disponía de otro proveedor extra educacional (ver Figura 4.1), que además de los sistemas mencionados anteriormente también incluye otro sistema cuántico real de 7 qubits (ibmq_casablanca) y permite, entre otras cosas, enviar una cantidad mayor de circuitos a un sistema o hacer más repeticiones en las ejecuciones. Se pueden ver todos los sistemas disponibles y características más detalladas como el tipo de procesador y su estructura en la página de servicios de IBM Q [26].

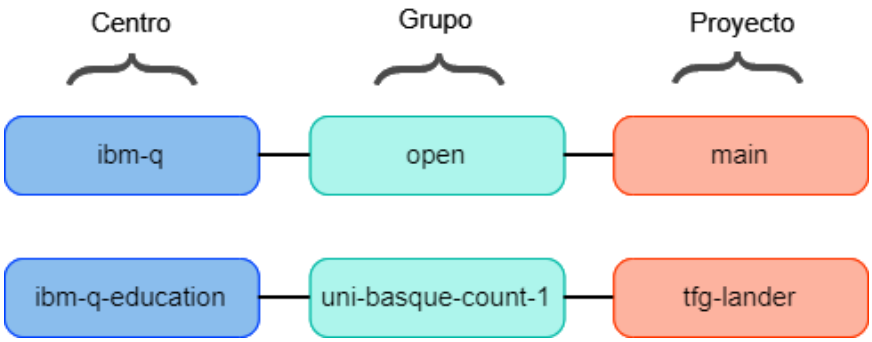


Figura 4.1 - Proveedores IBM Q

Como curiosidad adicional, se quiere mencionar que los nombres que IBM utiliza para sus sistemas cuánticos reales (manila, bogota…) no representan donde se ubican físicamente estos sistemas, sino que son nombres de ciudades donde se ubican las oficinas de IBM alrededor del mundo.

4.1.2. Envíos y cola de trabajos

Cuando se ejecuta una tarea utilizando un servicio de IBM Q como el envío de un circuito cuántico a un simulador o a un sistema real, se crea un trabajo y se obtiene una referencia de él (en el caso de ser ejecutado desde un código sería un objeto de Python). Con esta referencia es posible ver el progreso del envió de la tarea y una vez finalizada obtener los resultados finales de la computación.

Sin embargo, al enviar un trabajo a un sistema cuántico de IBM Cloud no se ejecuta directamente, primero se ingresa en una cola con los trabajos de otros usuarios antes de su ejecución. Para que ningún proveedor monopolice el uso de un sistema, el orden en el que se ejecutan es determinado mediante una fórmula de cola de reparto equitativa [27]. Este algoritmo intenta equilibrar la carga de trabajo entre diferentes proveedores para garantizar que cada uno obtenga la cantidad de tiempo asignada en el sistema.

Esto significa que los trabajos de distintos proveedores se pueden entrelazar sin importar el orden, por lo que los trabajos no tienen por qué ser ejecutados en el orden en el que se enviaron. Por esta razón, no es posible garantizar cuándo se ejecutará un trabajo concreto, ya que, aunque un trabajo en un principio parezca que esté por delante de los demás, a causa del reparto equitativo es posible que se ejecute más tarde.

La cola de reparto equitativa funciona mediante cuotas, es decir, el tiempo del sistema que se asigna a un proveedor determinado. Los proveedores con la mayor cantidad de tiempo son los que tienen la máxima prioridad. Dicha prioridad ira disminuyendo a medida que el proveedor consuma su tiempo en un sistema concreto. En definitiva, entre los trabajos que se envían a un sistema, se ejecutará en orden desde el que más prioridad tiene hasta el que menos.

Para las ocasiones en los que se necesite que un programa se ejecute en un tiempo determinado sin tener en cuenta el reparto de colas equitativo, IBM Q permite programar una reserva para una hora concreta en un sistema. Sin embargo, esta característica solo está disponible para organizaciones que pertenezcan a *IBM Q Network*.

Dentro de esta característica, existen 2 tipos de reserva distintos; el modo prioritario y el modo dedicado. En ambos modos, todos los trabajos que se envíen al sistema con el proveedor que se haya hecho la reserva, pasarán a estar al principio de la cola si son enviados dentro del tiempo reservado.

En la Figura 4.2 se puede observar el envió de trabajos de distintos proveedores (representados por colores). En el caso de arriba, al no tener una reserva, los trabajos de cada proveedor se ejecutarán en el orden que decida el algoritmo del reparto de cola. Sin embargo, en el segundo caso, asumiendo que el proveedor verde ha hecho una reserva, sus trabajos se ejecutarán uno detrás de otro sin tener que pasar por el reparto normal.

Los trabajos que se envíen antes o después de la reserva pasarán al flujo normal del reparto de colas. En cambio, los trabajos enviados en la reserva empezarán a ejecutarse en cuanto termine el último trabajo en ejecución que esté en ese momento en el Backend.

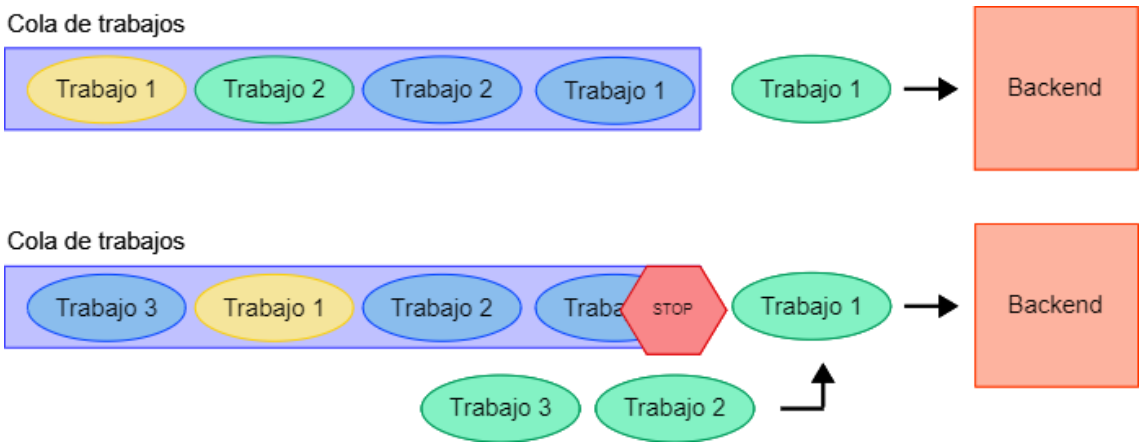


Figura 4.2 – Envió de trabajos sin reserva (arriba) y con reserva (abajo)

La diferencia entre el modo prioritario y el dedicado es que, en el primer caso, si no se envían trabajos dentro de la reserva realizada, el flujo normal continua, por lo que los demás trabajos se ejecutarán como si no hubiese una reserva. Sin embargo, en el modo dedicado, la cola de

trabajos se detiene completamente hasta finalizar la reserva, por lo que, aunque el proveedor que ha hecho la reserva no mande ningún trabajo, no se van a ejecutar los demás.

Como se ha mencionado en el apartado de proveedores, se disponía de un proveedor educacional, siendo parte de la red de IBM Q. Sin embargo, con este proveedor solamente ha sido posible hacer las reservas de modo prioritario en el sistema ibmq_casablanca.

4.1.3. Composer y Lab

Uno de los elementos que tiene IBM Q es Composer, una herramienta de programación cuántica gráfica que permite arrastrar y soltar distintas operaciones para formar circuitos cuánticos y ejecutarlos tanto en los simuladores como en los sistemas reales (ver Figura 4.3). A medida que se va creando el circuito, permite visualizar las probabilidades de colapso de los qubits.

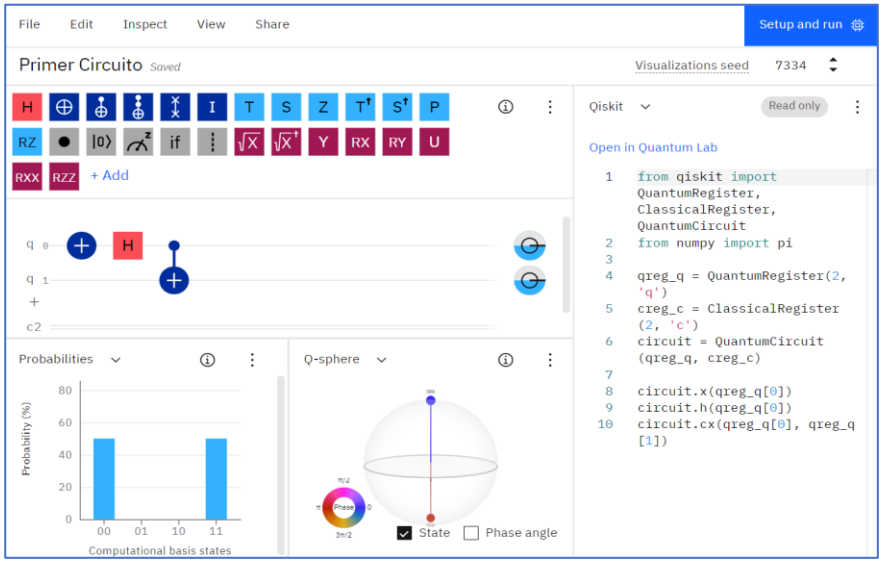


Figura 4.3 - IBM Q Composer

Por otro lado, también se va generando el código del circuito OpenQASM (lenguaje intermedio para instrucciones cuántica) o Qiskit y permite editarlo en cualquier momento reflejando los cambios en el circuito. Se puede encontrar más información sobre los componentes de Composer en su documentación [28].

Otra de las herramientas de IBM Quantum es Lab, un entorno de programación en la nube para construir aplicaciones cuánticas y experimentos con Qiskit [29]. Esta herramienta utiliza un entorno personalizado de Jupyter Notebook, por lo que es posible escribir código, ecuaciones y textos narrativos de forma combinada y ejecutar el código en sistemas reales o en simuladores (ver Figura 4.4).

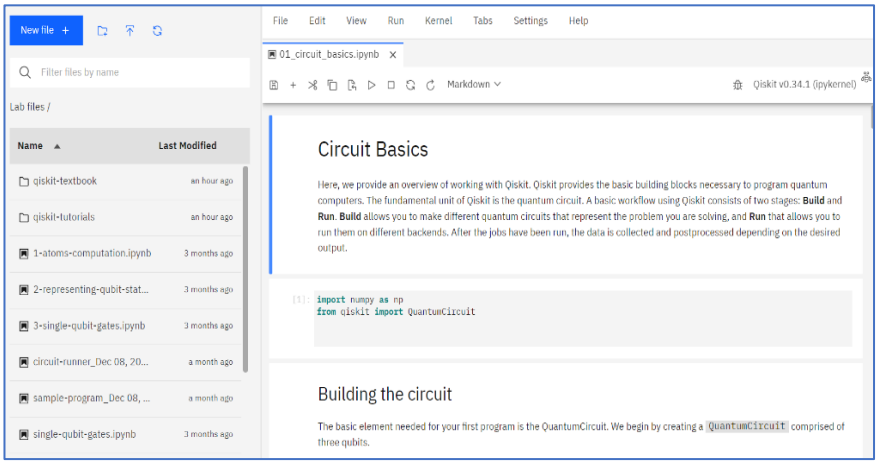


Figura 4.4 - IBM Q Lab

La mayor ventaja de esta herramienta es que como es un servicio en la nube, se puede acceder desde cualquier navegador y continuar con el trabajo en cualquier momento sin necesidad de instalar nada. Además, contiene todos los cuadernos de Qiskit con tutoriales o aplicaciones que se pueden abrir y modificar.

4.2. Qiskit

Como ya se ha mencionado, Qiskit es el kit de desarrollo software de código abierto para trabajar con computadoras cuánticas a nivel de circuitos, pulsos y algoritmos creado por IBM. Adicionalmente, permite crear y manipular programas cuánticos y ejecutarlos en dispositivos cuánticos reales de IBM o en simuladores tanto de IBM como en una computadora local [18].

Qiskit es una pila de software que facilita a cualquier persona el uso de computadoras cuánticas, independientemente de su nivel de habilidad o área de interés, permitiendo diseñar experimentos y aplicaciones fácilmente y poder ejecutarlos para obtener sus resultados. Este software ya está siendo utilizado en todo el mundo por principiantes, aficionados, educadores, investigadores y empresas comerciales y actualmente lo siguen actualizando para añadir o renovar algunas de sus funcionalidades.

Qiskit usa el lenguaje de programación Python y se puede instalar e importar como una librería para posteriormente integrarlo y utilizarlo con Jupyter Notebook. Sin embargo, como se ha visto en el apartado de IBM, esto no es un requerimiento obligatorio, ya que es posible trabajar con él en la nube con IBM Q Lab.

En las siguientes secciones se analizan los elementos sobre los que se basa Qiskit además de analizar el funcionamiento de transpilar los circuitos.

4.2.1. Elementos

Qiskit proporciona la capacidad de desarrollar software cuántico tanto a nivel de código de máquina de OpenQASM como a niveles abstractos adecuados para usuarios finales sin experiencia en computación cuántica. Estás funcionalidades son proporcionadas por los elementos que lo componen; Terra, Ignis y Aer [30].

En primer lugar, se encuentra el elemento **Terra**, el cual es la base sobre la que trabajan los demás elementos de Qiskit. Terra proporciona las herramientas para componer programas

cuánticos a nivel de puertas cuánticas y de pulsos. En esta capa también se lleva a cabo la optimización de los programas según las limitaciones de un dispositivo en particular y gestiona la ejecución de lotes de experimentos en dispositivos de acceso remoto, es decir, controla la comunicación entre el entorno de ejecución del programa y el usuario.

Uno de sus módulos principales es el transpilador, el cual es el encargado de poder ejecutar circuitos cuánticos en dispositivos reales. Los ordenadores cuánticos reales solo pueden ejecutar un conjunto de puertas cuánticas calibradas físicamente y específicas del hardware, por lo cual, se requiere transformar las puertas definidas por humanos a puertas compatibles con la máquina, optimizando el rendimiento donde sea posible.

Dentro de Terra también se encuentra el acceso a los proveedores de un usuario y permite seleccionar el backend al que enviar un trabajo concreto para su ejecución, ya sea un simulador o un sistema real. Una vez enviado el trabajo, permite acceder a la información de este para comprobar en qué estado se encuentra (en ejecución, en cola…) y una vez finalizado, obtener los resultados de la ejecución.

Este elemento también contiene herramientas para la visualización, permitiendo por ejemplo visualizar un circuito creado para comprobar si es realmente el que se estaba diseñando, o una vez recuperados los resultados de un circuito ejecutado, visualizar en un histograma las probabilidades obtenidas de cada estado.

Otro de los elementos con los que cuenta Qiskit es **Aer**. Este elemento proporciona simuladores de computación cuántica de alto rendimiento con modelos de ruido realistas que permiten acercar lo máximo posible la simulación de un programa cuántico a la ejecución en un entorno real. Qiskit Aer incluye tres backends de simuladores; QasmSimulator (permite la ejecución ideal y ruidosa asemejándose lo máximo posible a un entorno real), StateVectorSimulator (permite la ejecución ideal de un circuito devolviendo el vector de estado final del simulador después de la aplicación) y UnitarySimulator (permite la ejecución ideal de una sola repetición de circuitos devolviendo una matriz unitaria final del circuito en sí).

El tercer elemento de Qiskit es **Ignis,** el cual proporciona herramientas para la verificación de hardware cuántico, la caracterización de ruido y la corrección de errores. Este elemento está destinado principalmente a aquellos que deseen diseñar códigos de corrección de errores cuánticos.

Sin embargo, desde la versión 0.7 de Ignis, se encuentra obsoleto y ha sido reemplazado por el proyecto Qiskit Experiments. Por lo tanto, el desarrollo activo del proyecto se ha detenido y algunos de sus funcionalidades han sido migrados al elemento Terra y a Experiments [31].

Por último, Qiskit utilizaba un cuarto elemento llamado **Aqua,** el cual proporcionaba una biblioteca de algoritmos que permitían crear aplicaciones orientados en distintos ámbitos como la optimización, finanzas, aprendizaje automático y naturaleza. Sin embargo, desde la salida de la versión 0.9 con el lanzamiento 0.25.0 Qiskit, este elemento se ha reestructurado dividiéndose en módulos individuales según su aplicación.

Estos nuevos módulos son **Qiskit Optimization**, **Qiskit Finance**, **Qiskit Machine Learning** y **Qiskit Nature**. Otros algoritmos y elementos principales también se han movido dentro del elemento Qiskit Terra. Se puede encontrar más información sobre la migración en el apartado de guía de migración de Qiskit Aqua [32].

En el caso de **Qiskit Optimization**, permite el modelado de alto nivel de problemas de optimización, con conversión automática de problemas a diferentes representaciones requeridas hasta un conjunto de algoritmos y optimización cuántica fáciles de usar que están listos para ejecutarse en simuladores clásicos, así como en dispositivos cuánticos reales a través de Qiskit.

En cuanto a **Qiskit Finance**, este contiene componentes de incertidumbre para problemas de acciones y valores, traductores Ising para problemas como la optimización de cartera y proveedores de datos para obtener datos reales o aleatorios para experimentos financieros.

Por otro lado, **Qiskit Machine Learning** contiene algoritmos de clasificación como QSVM y QSVC además de distintos conjuntos de datos con los que poder utilizar dichos clasificadores. También contiene herramientas para la implementación de algoritmos de redes neuronales cuánticas y permite integrarlos en un flujo de trabajo de PyTorch (una biblioteca de aprendizaje automático).

Por último, **Qiskit Nature** proporciona varias funcionalidades para experimentar con la computación cuántica en problemas de ciencias naturales, como la química y física. Por ejemplo, permite calcular la energía del estado fundamental o la energía de las moléculas en un estado de excitación.

4.2.2. Transpilación

Cuando se crea un circuito cuántico, los sistemas como Qiskit permiten a los usuarios ejecutar estos circuitos sin tener que saber realmente el funcionamiento interno que tienen los sistemas cuánticos reales. Sin embargo, cada sistema está construido de una forma distinta, y los qubits que los componen pueden estar físicamente conectados de diversas formas.

Estas conexiones físicas entre los qubits de los sistemas reales se representan con los coupling map o mapas topológicos, y representan cómo están conectados los qubits entre sí. En la Figura 4.5 se pueden ver los mapas topológicos de 2 de los sistemas de IBM Q (ibmq_belem y ibmq_casablanca).

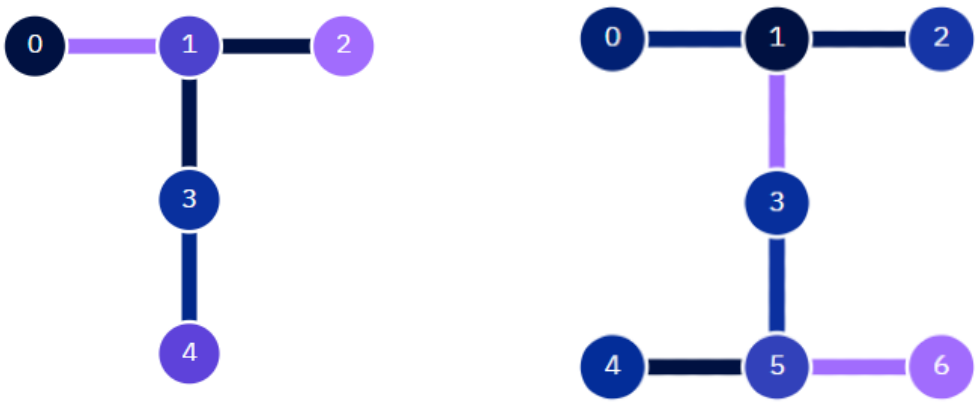


Figura 4.5 - Mapas topológicos de ibmq_belem y ibmq_casablanca

Tomando como ejemplo el mapa topológico de ibmq_belem, podemos expresar ese grafo de la siguiente manera: [[0,1], [1,0], [1,2], [2,1], [1,3], [3,1], [3,4], [4,3]]. En el caso de [0,1], [1,0], esto nos indica que el qubit 0 (q0) y el qubit 1 (q1) están conectados, al contrario que q0 y q3 que no están conectados.

A primera vista, esto puede hacer pensar que no es posible hacer ciertas operaciones, ya que por ejemplo no se podría realizar una puerta controlada CX entre q0 y q3 por que no están directamente conectados.

Sin embargo, Qiskit consigue abstraer todo esto consiguiendo que aparentemente para nosotros todos los qubits estén conectados con los demás. Lo que realmente ocurre, son ciertas transformaciones para poder ejecutar los circuitos que le indiquemos.

Como un ejemplo, podemos ver en la Figura 4.6 que para conseguir ejecutar la puerta CX entre los qubits 0 y 3, se utiliza una puerta Swap para intercambiar de lugar los qubits, el cual se descompone en cuatro puertas CNOT, lo que representa una operación costosa y ruidosa de realizar.

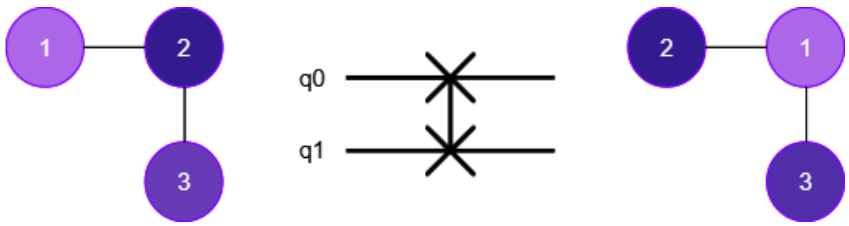


Figura 4.6 - Reubicación de los qubits

Por ello, para lo que aparentemente para nosotros era utilizar una puerta CX entre 2 qubits, internamente se han utilizado 4 puertas en total (ver Figura. 4.7). Aunque este ha sido un ejemplo pequeño, cada puerta añadida aumenta el error que se genera en las computadoras cuánticas, por lo que en circuitos más avanzados en los que se utilicen este tipo de puertas, el error generado puede ser bastante considerable.

Figura 4.7 - Puertas utilizadas para reubicar un qubit

Por otro lado, también hay que tener en cuenta las puertas que tienen disponibles los sistemas cuánticos reales. Cuando desarrollamos un circuito utilizamos un conjunto de puertas cuánticas abstractas, independientes del dispositivo, para representar y resolver un problema. Sin

embargo, los dispositivos cuánticos reales solo pueden ejecutar un conjunto limitado de puertas cuánticas calibradas físicamente y específicas del hardware. Por lo tanto, es necesario la traducción de las puertas que definimos en nuestros circuitos a puertas compatibles con la máquina, optimizando el rendimiento cuando sea posible.

Dentro de IBM Q, se puede observar cuales son las puertas disponibles para cada sistema. Tomando como ejemplo el sistema ibmq_belem, este dispone de las puertas básicas CX, ID, RZ, SX y X. En la Figura 4.8 se puede observar un circuito cuántico antes de ser ejecutado en el sistema, y justo debajo la transformación que se realiza para adecuar las puertas indicadas en el circuito original a las puertas que pueden ser utilizadas en el sistema enviado.

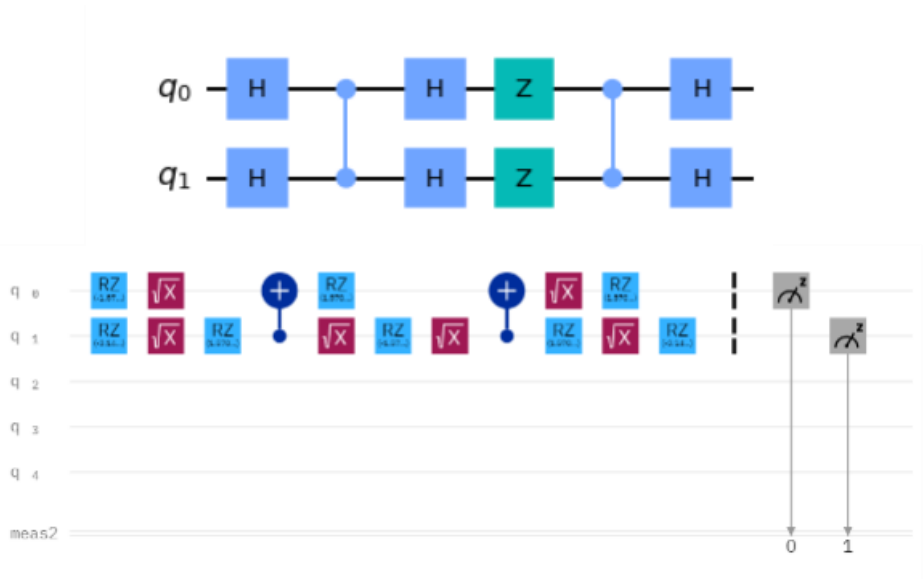


Figura 4.8 - Circuito antes de ejecutarse(arriba) y tras la transpilación (abajo)

Estos procesos de adecuar nuestras puertas cuánticas a las que pueden utilizar los sistemas reales, junto con el proceso de asignar los qubits virtuales de los circuitos a qubits físicos dentro del sistema es parte de la traspilación que ocurre dentro del elemento Terra de Qiskit [33].

En definitiva, la transpilación es el proceso encargado de reescribir un circuito de entrada dado para que coincida con la topología de un dispositivo cuántico específico y para optimizar su ejecución en los ruidosos sistemas cuánticos actuales.

Aunque este proceso se realice automáticamente al ejecutar un circuito, sí que es posible cambiar algunos parámetros como el nivel de optimización que se desea, y también sería recomendable ejecutar los circuitos en distintos sistemas y ver en cual se adecuaría mejor. Sin embargo, para este trabajo de investigación esto no se ha tenido en cuenta y a la hora de utilizar sistemas reales se han utilizado los menos ocupados a nivel de trabajos en ese momento.

4.2.3. Runtime

Una de las nuevas funcionalidades añadidas para Qiskit este mismo año ha sido Runtime, una nueva arquitectura ofrecida por IBM Q que agiliza los cálculos que requieren muchas iteraciones [34]. El objetivo de esta nueva arquitectura es reducir notablemente el exceso en tiempo de computación cuando se envían aplicaciones y algoritmos a procesadores cuánticos que

requieren muchas iteraciones de ejecuciones de circuitos y que también necesitan un procesamiento clásico (ver Figura 4.9).

En general, un programa de Qiskit Runtime, es una pieza de código de Python que toma ciertas entradas, realiza computación cuántica y tal vez clásica, proporciona de forma interactiva resultados intermedios si se desea y devuelve los resultados del procesamiento.

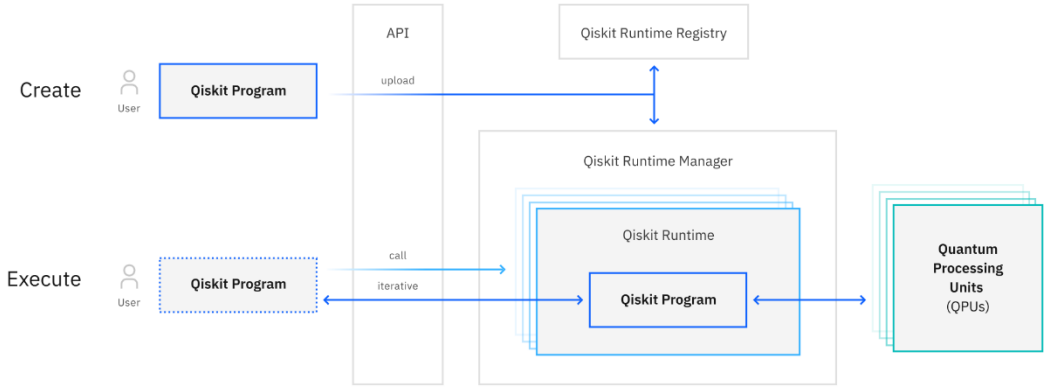


Figura 4.9 - Arquitectura Qiskit Runtime [35]

Por lo tanto, el objetivo de utilizar Runtime es reducir la latencia que se genera a medida que el código pasa entre el dispositivo de un usuario y la computadora cuántica basada en la nube, permitiendo a los desarrolladores ejecutar su programa en el entorno de ejecución Qiskit Runtime, donde la nube híbrida de IBM manejará todo el flujo.

Con esta arquitectura, los desarrolladores de IBM Q han llegado a obtener una mejora hasta del 120% en cuanto a velocidad en simulaciones de moléculas [36] y actualmente se encuentra disponible para poder utilizarlo por todos los proveedores en cualquiera de sus sistemas, ya sea en simuladores o máquinas reales.

5. APLICACIONES

La computación cuántica puede ser utilizada en distintos ámbitos, desde la criptografía, problemas de optimización o hasta algoritmos cuánticos para resolver problemas concretos que prometen una mejora considerable respecto a sus versiones clásicas. En esta sección, se analizan algunas de estas aplicaciones y se expone la teoría general de ellos, mientras que en la siguiente sección se analizan los experimentos realizados para probar dichas aplicaciones.

Uno de los algoritmos cuánticos que prometen una mejora cuadrática en tiempo respecto a una solución clásica es el algoritmo de Grover, utilizado para la búsqueda en una secuencia no ordenada de N elementos. Este será el algoritmo cuántico que se analizará para comprobar si realmente se obtiene esa mejora y si es adaptable para utilizarlo en otro tipo de problemas.

Por otro lado, para analizar aplicaciones de la computación cuántica en distintos ámbitos, se ha basado en los módulos de Qiskit de Machine Learning, Optimization y Finance, y en cada uno de ellos se ha experimentado con algunos de los problemas que permiten resolver.

Dentro del módulo de Machine Learning, se ha probado la implementación del algoritmo QSVM y QSVC, que pueden ser utilizados a la hora de entrenar un clasificador para posteriormente utilizarlos para clasificar un nuevo conjunto de datos.

En cuanto al módulo de Optimización de Qiskit, se ha analizado su usabilidad para modelar problemas de tipo QUBO y después poder resolverlas con el algoritmo QAOA. Para ello, se ha analizado el problema Max-Cut el cual se explica en las siguientes secciones.

Por último, dentro del módulo de finanzas de Qiskit se ha analizado la carga de distintos conjuntos de datos y el algoritmo VQE para resolver un problema de tipo Portfolio Optimization.

A continuación, se profundiza más en cada una de las aplicaciones y se expone su teoría general. Como en este trabajo no se ha centrado únicamente en un problema si no que se ha querido probar en distintos ámbitos, no se ha profundizado demasiado en cada algoritmo y no se van a analizar todos los parámetros que los componen.

5.1. Algoritmo de Grover

Este algoritmo permite hacer una búsqueda en una secuencia no ordenada de datos en un tiempo cuadrático menor al que lo conseguiría una computadora clásica. Tanto la teoría como la implementación del algoritmo se ha obtenido a partir de la documentación de Qiskit [37].

Supongamos que tenemos una lista desordenada con N elementos (ver Figura 5.1). En esta lista el objetivo es encontrar el elemento X (marcado en rojo). Utilizando la computación clásica, el coste de encontrar este elemento sería $\frac{N}{2}$ de media, y en el peor de los casos N, ya que tendría que recorrer uno a uno todos los elementos y ver en cada caso si es el elemento que buscamos o no.

En cambio, en la computación cuántica, utilizando la amplificación de la amplitud que se explica a continuación, se puede encontrar el elemento deseado con un coste aproximado de $\sqrt{N}$, lo cual es una mejora notable respecto a su versión clásica.

Suponiendo que coger cada elemento para observarlo nos lleva un segundo, en una lista de 100 elementos, en el peor de los casos nos llevaría 100 segundos encontrar ese elemento con la solución clásica. En cambio, con el algoritmo cuántico de Grover, esta misma operación nos llevaría tan solo 10 segundos aproximadamente.

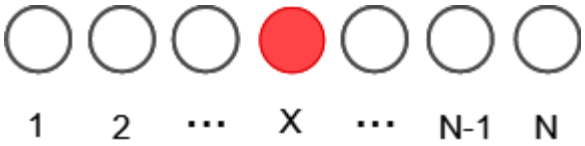


Figura 5.1 - Lista desordenada de N elementos

La idea general de los pasos que sigue el algoritmo de Grover es el siguiente:

1. Poner todos los qubits en estado de superposición para obtener todas las posibles combinaciones de la variable de entrada.
2. Implementar un oráculo que marque el elemento que se desea encontrar. Este oráculo funciona invirtiendo la fase de la amplitud del elemento que se busca.
3. Implementar un difusor, es decir, el circuito de amplificación que incremente la amplitud del elemento marcado mientras que decrementa la amplitud de los otros estados
4. Medir todos los qubits. El elemento marcado obtiene una probabilidad muy alta de colapsar ya que su amplitud ha sido amplificada.

El objetivo del oráculo es marcar el elemento deseado con una fase negativa. Por ejemplo, para un estado cualquiera $|\varphi\rangle$ el oráculo U_f se podría expresar con la siguiente función, poniendo una fase negativa al elemento φ en el caso de que sea el elemento x que buscábamos:

$$U_f|\mathrm{x}\rangle = \begin{cases} |\varphi\rangle \text{ si } \varphi \neq \mathrm{x} \\ -|\varphi\rangle \text{ si } \varphi = \mathrm{x} \end{cases}$$

Para el primer paso se consigue una superposición uniforme de todos los estados. Para ello, se aplica la puerta Hadamard en todos los qubits: $|\psi_0\rangle = \mathrm{H}^{\otimes n}|0\rangle^n$. En este primer paso, todos los estados se encuentran en una superposición uniforme, con la misma probabilidad de colapsar hacia su estado (ver Figura 5.2).

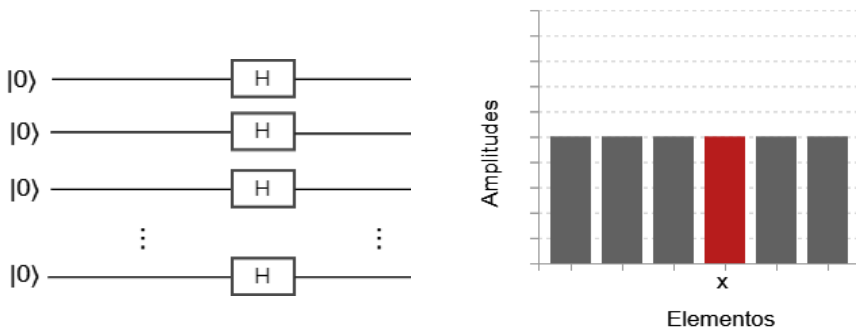


Figura 5.2 - Estado Inicial con todos los elementos en estado de superposición

Una vez que se tienen todos los elementos en un estado de superposición, se aplica el oráculo U_f para marcar el elemento que se quiere encontrar. Como se puede ver en la Figura 5.3, el elemento x que buscábamos queda con una amplitud negativa. En este estado todavía no nos sirve medir los qubits, ya que la probabilidad de colapsar hacia un estado, al obtenerse con el módulo al cuadrado, el valor negativo se convertiría en positivo y seguiríamos teniendo las mismas probabilidades en todos los elementos.

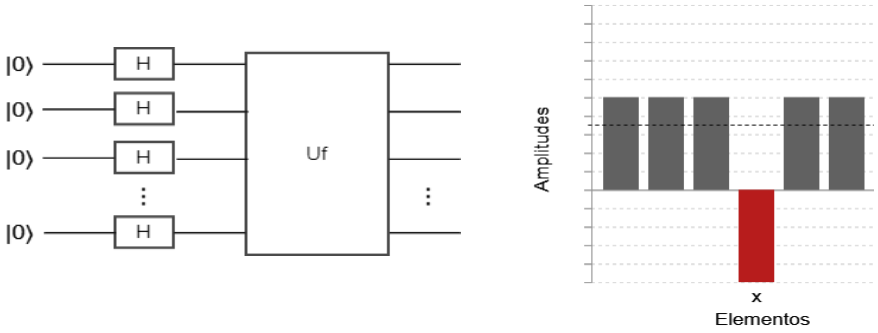


Figura 5.3 - Estado después de aplicar el oráculo

En el último paso, se aplica una rotación con el difusor U_s en el estado$|\psi_1\rangle$, consiguiendo que la amplitud negativa del elemento que buscábamos se amplifique superando a los demás elementos. Es decir, se hace una simetría de todos los estados respecto al de la media (ver Figura 5.4).

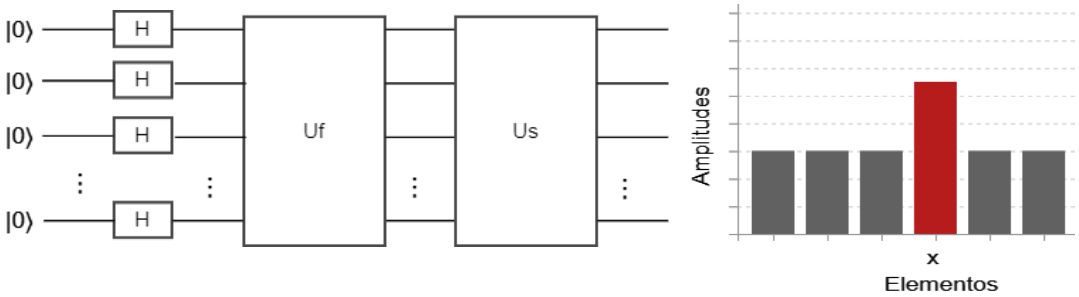


Figura 5.4 - Estado final con amplificación de fase en el elemento que se buscaba

Dado que la amplitud promedio se había reducido por la primera reflexión, esta transformación aumenta la amplitud negativa de $|x\rangle$ a aproximadamente tres veces su valor original, mientras que disminuye las otras amplitudes. Después, se repite el procedimiento desde el paso 2 para seguir aumentando la probabilidad del elemento que deseamos.

Es decir, como se ve en la figura 5.5, se repite el circuito U_f y U_s 2 veces, lo cual es lo que se denomina 2 iteraciones en el algoritmo de Grover. Por lo tanto, con 3 iteraciones, se añadiría otro bloque más que implemente el mismo oráculo y difusor, y así sucesivamente con las iteraciones que se deseen.

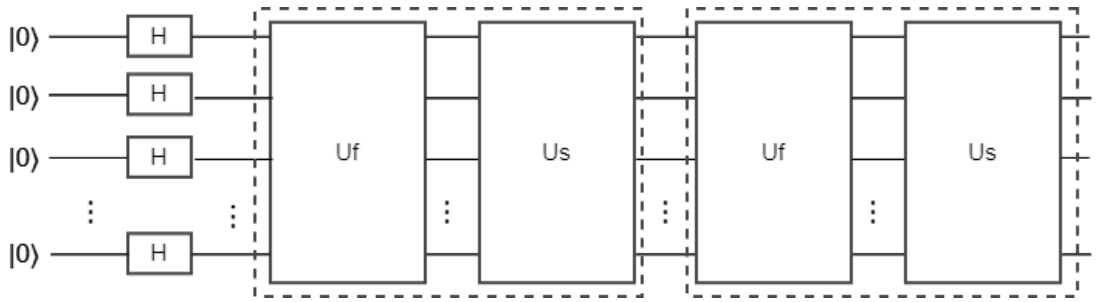


Figura 5.5 - Iteraciones en el algoritmo de Grover

Sin embargo, tal y como se va a ver en la parte de experimentación, a partir de cierta cantidad de iteraciones se va reduciendo la probabilidad de obtener el estado deseado.

5.2. Machine Learning

Para resolver un problema en una computadora se hace uso de un algoritmo. Un algoritmo no es más que una secuencia de instrucciones que deben llevarse a cabo para transformar la entrada a la salida. Por ejemplo, se puede idear un algoritmo de ordenación en el cual la entrada es un conjunto de números y la salida es el mismo conjunto, pero ordenado. Estas tareas pueden realizarse con distintas implementaciones de algoritmos por lo que la mejor opción sería obtener el más eficiente o el que requiera el menor número de instrucciones [38].

Sin embargo, para otras tareas como un detector de correos maliciosos o mensajes de spam no se dispone directamente de un algoritmo, por lo que es aquí cuando se utilizaría el Machine Learning o el aprendizaje automático. Esto es un método de análisis de datos que automatiza la construcción de modelos analíticos, siendo parte de la rama de la inteligencia artificial basada en la idea de que los sistemas pueden aprender de datos, identificar patrones y tomar decisiones con mínima intervención humana [39].

En otras palabras, mediante el aprendizaje automático se desea que la computadora extraiga automáticamente el algoritmo para una tarea concreta.

En el aprendizaje automático existen diferentes tipos de algoritmos dependiendo de la salida que ofrezcan, como la clasificación supervisada, clasificación no supervisada, aprendizaje por refuerzo… Para este trabajo, los algoritmos analizados han sido los de aprendizaje supervisado.

Para el caso del aprendizaje supervisado, se dispone de un conjunto de datos llamados instancias, los cuales vienen representados mediante una serie de variables predictoras y cada una de ellas tiene asociada una clase o un grupo al que pertenece (por ejemplo, spam o no spam). El objetivo es estimar un modelo de clasificación capaz de predecir la clase para un nuevo elemento dado (ver Figura 5.6).

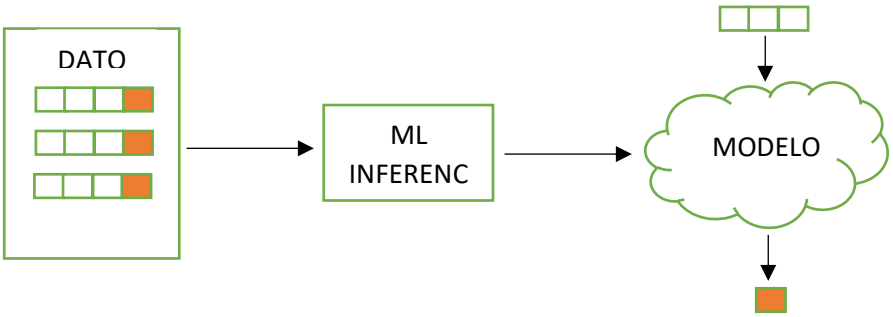


Figura 5.6 - Aprendizaje Supervisado

Como ejemplo, siguiendo con el caso de los mensajes de spam o no spam, se dispondrían como datos de entrada todos los mensajes con el que entrenar el clasificador. Estos mensajes, después de transformarlos a la forma de variables, se le pasan a un clasificador, el cual será entrenado creando un modelo. Una vez que ya se tiene el modelo, cuando llegue un nuevo mensaje, se podrá estimar automáticamente que tipo de mensaje es; spam o no spam.

A la hora de analizar un conjunto de datos, cada instancia puede estar representadas por una gran cantidad de variables, sin embargo, en ocasiones puede ser necesario reducir la dimensionalidad de estas para poder visualizar o probar a entrenar los clasificadores.

Para esta reducción de dimensionalidad una de las técnicas que se utilizan en al aprendizaje automático es el análisis de componentes principales (Principal Component Analysis PCA). Este método permite reducir la dimensionalidad para simplificar la complejidad de espacios con múltiples dimensiones a la vez que conserva su información [40].

En un conjunto de n observaciones o instancias, las cuales están representadas por *p* variables ($X_1, X_2, \dots, X_p$), el algoritmo PCA permite encontrar un número menor de componentes *z* (z < p) las cuales mantienen lo máximo posible de la información de las *p* variables originales. Por lo tanto, el conjunto que antes estaba caracterizado por p variables, ahora es posible describirlo con *z* variables, cada una de las cuales recibe el nombre de componente principal.

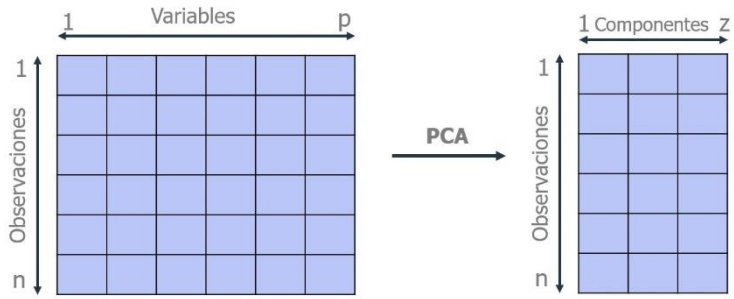


Figura 5.7 - Reducción de dimensiones con PCA [40]

5.2.1. SVM y QSVM

Como se ha mencionado, existen diferentes algoritmos e implementaciones para una sola tarea, al igual que existen distintos tipos de clasificadores. Uno de ellos, y el que se va a analizar a continuación, son los **Suport Vector Machine** (SVM) y su implementación cuántica de Qiskit **Quantum Support Vector Machine** (QSVM).

Support vector machine es un algoritmo de aprendizaje supervisado que se utiliza en muchos problemas principalmente de clasificación, aunque también puede ser utilizado en regresión. Algunas de sus aplicaciones pueden ser las médicas de procesamiento de señales, procesamiento del lenguaje natural y reconocimiento de imágenes y voz [41].

En un conjunto de datos, puede haber muchos hiperplanos que separen las clases de forma correcta, pero el objetivo del algoritmo SVM es encontrar un hiperplano que maximice la distancia entre los puntos de datos de distinta clase de la mejor forma posible. Estos puntos son los que se denominan vectores de apoyo, porque son las observaciones de datos que ayudan a determinar el límite de decisión (ver Figura 5.8).

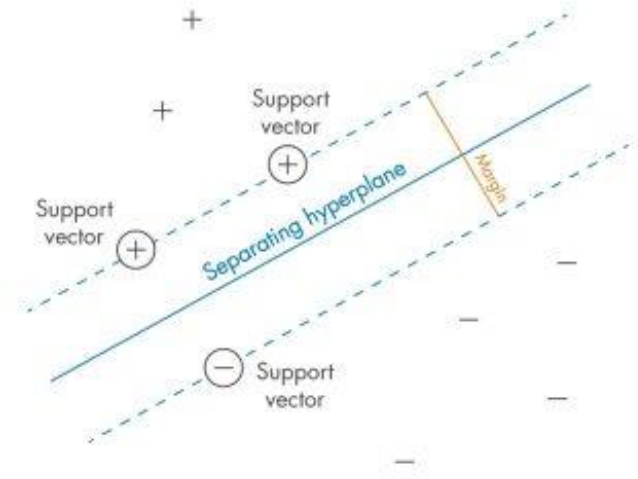


Figura 5.8 – SVM [41]

El algoritmo solo puede encontrar este hiperplano en problemas que permiten separación lineal, y en la mayoría de los problemas prácticos, el algoritmo maximiza el margen flexible permitiendo un pequeño número de clasificaciones erróneas. Una vez que se ha entrenado el clasificador encontrando el hiperplano de margen máximo, dependiendo de qué lado del hiperplano se ubique un nuevo punto de datos, podríamos asignar a la observación la clase que corresponda a ese lado del hiperplano.

Aunque en un principio parezca una solución sencilla, en la práctica la mayoría de las veces los datos no son linealmente separables, ya que los datos se distribuyen aleatoriamente, lo que dificulta la separación de diferentes clases de forma lineal. Sin embargo, se puede transformar el conjunto de datos de su espacio dimensional original a un espacio dimensional mayor. Con esto, podemos conseguir que las clases ahora sean linealmente separables en este espacio de características dimensional superior, por lo que podríamos encontrar un hiperplano que divida claramente los datos de las diferentes clases (ver Figura 5.9).

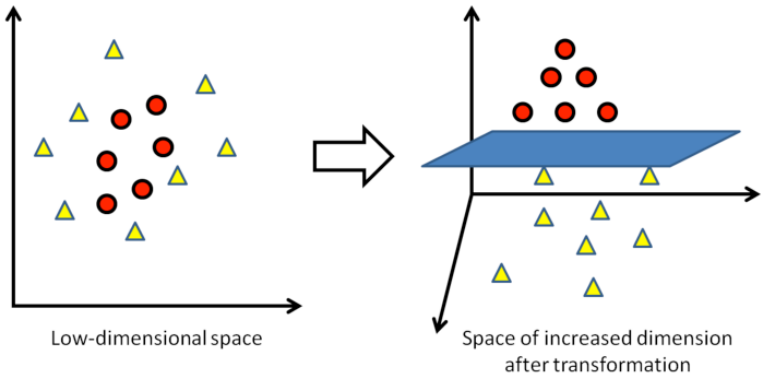


Figura 5.9 - SVM Cambio de dimensionalidad [79]

Este cambio de dimensionalidad se logra con las funciones kernel, asignando los datos a un espacio dimensional diferente, que suele ser superior, con la expectativa de que resulte más fácil separar las clases después de esta transformación.

En este proceso, los datos no se tienen que transformar explícitamente, lo que supondría una alta carga computacional. Aquí es cuando se utiliza lo que se denomina truco de kernel. Nos permite operar en el espacio de características original sin calcular las coordenadas de los datos en el espacio dimensional superior.

La función de kernel se puede definir como: $k(\vec{x}_i,\vec{x}_j) = \langle f(\vec{x}_i), f(\vec{x}_j)\rangle$ Donde k es la función kernel, $\vec{x}_i,\vec{x}_j$ son las entradas de n dimensiones y f es una función de mapeo desde n-dimensiones hasta m dimensiones y $\langle a, b\rangle$ denota el producto interno. Cuando se consideran datos finitos, una función de kernel se puede representar como una matriz: $K_{ij} = k(\vec{x}_i,\vec{x}_j)$

En computación cuántica, se utiliza un mapa de características cuánticas $\phi(\vec{x})$ para asignar un vector de características clásicas $\vec{x}$ a un estado cuántico $|\phi(\vec{x})\rangle\langle\phi(\vec{x})|$ tal que $K_{ij} = \langle\phi^{\dagger}\ (\vec{x}_j)|\phi(\vec{x}_i)\rangle|^2$ [42].

Como ejemplo, en la Figura 5.10 se puede observar la representación de un mapa de características en un qubit. Un conjunto de datos que se encuentra en el intervalo $\Omega = (0,2\pi]$ con una clase binaria (+1, -1) puede ser mapeado a una esfera de Bloch (líneas rojas y azules) utilizando un mapa de características no lineal [43], donde $U\phi(\vec{x})$ es una puerta cuántica de fase de ángulo $x \in \Omega$.

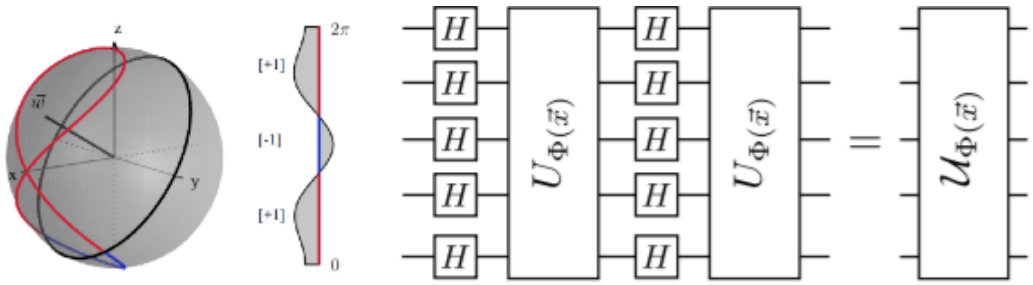


Figura 5.10 - Mapa de características [43]

5.3. Optimización

Otro de los módulos que se ha probado ha sido el de optimización. La optimización trata de buscar una solución óptima entre un conjunto finito de soluciones posibles. Está definido respecto a un criterio de función, llamado función objetivo, el cual se tiene que minimizar o maximizar.

Entre los problemas de minimización se encuentran problemas como minimizar la distancia, el coste, el material empleado en un proceso, o el consumo de energía. Como ejemplo, un problema de minimización puede ser aquel que encuentre el camino más corto entre dos puntos A y B o el que minimice los costes de producción de un producto.

En cambio, en problemas de maximización, tal y como lo indica su nombre, el objetivo es maximizar un valor. Como ejemplo, un problema de maximización puede ser aquel que busque el mayor precio de venta, la eficiencia o el número de productos producidos.

En cuanto al módulo Qiskit Optimization, este encubre toda la gama de resolución de problemas de optimización, desde el modelado de alto nivel de este tipo de problemas hasta su conversión automática a la representación necesaria para poder solucionarlos con distintos algoritmos.

El módulo tiene soporte para programas cuadráticos restringidos cuadráticamente (Quadratically constrained quadratic program QCQP), es decir, problemas de optimización cuya función objetivo y restricciones son funciones cuadráticas (el grado máximo de las variables es dos).

Matemáticamente, un programa cuadrático se puede describir de la siguiente forma:

$$\begin{aligned} &minimize\ x^T Q_0 x + \ c^T x \\ &subject\ to\ Ax \leq b \\ &\qquad x^T Q_i x + \ a_i^T x \leq r_i, 1, \dots, i \dots, q \\ &\qquad li \leq x_i \leq u_i, 1, \dots, i, \dots, n \end{aligned}$$

En esta expresión, $\mathrm{Q_i}$ son matrices de dimensionalidad n x n, *A* es una matriz de dimensionalidad m x n, *x* y *c* son vectores de dimensionalidad n, *b* es un vector de dimensionalidad m y, por último, *x* puede ser definido como una variable binaria, entera o continua [44]. Como ejemplo, un programa cuadrático puede ser el siguiente:

$$\begin{aligned} &minimize\ 3x_1x_2 + x_2 - 2x_3 \\ &subject\ to\ x_1 + x_2 + x_3 = 2 \\ &\qquad x_1, x_2, x_3 \in \{0,1\} \end{aligned}$$

En este ejemplo, la función objetivo que se desea minimizar es $3x_1x_2 + x_2 - 2x_3$ y las restricciones que se deben cumplir son $x_1 + x_2 + x_3 = 2$ y que las 3 variables sean binarias.

A pesar de que este ha sido un ejemplo muy simple, los problemas cuadráticos de este tipo cubren una gran variedad de problemas y Qiskit puede trabajar con ellas, con problemas como Max-Cut, el problema del viajante (Travelling salesman problem), problema de la mochila (Knapsack problem) o el problema de enrutamiento de vehículos (Vehicle routing problem), entre otros [45].

Para crear un modelo de este tipo, Qiskit dispone de una clase Python llamado *QuadraticProgram.* Este objeto permite generar directamente problemas cuadráticos especificando las variables a utilizar, las restricciones y la función objetivo a maximizar o minimizar. También permite cargar los problemas desde otras librerías de optimización clásicas o desde ficheros LP (un fichero de formato común para problemas de optimización).

Sin embargo, los algoritmos cuánticos no pueden trabajar directamente con este tipo de problemas, es necesario convertirlos primero a problemas cuadráticos de optimización binaria sin restricciones (Quadratic unconstrained binary Optimization, QUBO). Es decir, se trata de un problema cuadrático sin restricciones:

$$\begin{aligned} &minimize\ \Sigma_{ij} c_{ij} x_i x_j + \ \Sigma_{\mathrm{i}} \mathrm{d_i x_i} \ + \ \mathrm{M}\, \Sigma_{\mathrm{j}} \left(\mathrm{b_j} - \ \Sigma_{\mathrm{i}} \mathrm{a_{i,j} x_i}\right)^2 \\ &subject\ to\ x_i \in \{0,1\} \end{aligned}$$

Siguiendo con el ejemplo anterior del programa cuadrático con restricciones, su versión QUBO sería la siguiente:

$$minimize\ 3x_1x_2 + x_2 - 2x_3 + M\ (x_1 + x_2 + x_3 - 2)^2$$
$$subject\ to\ \ x_1, x_2, x_3 \in \{0,1\}$$

Como se puede ver, se obtiene el mismo problema cuadrático, pero añadiendo las restricciones dentro de la función a optimizar. También se añade la variable numérica *M* que toma un valor muy grande, por lo que, si no se cumple la restricción, penaliza la función a minimizar sumándole un valor grande.

Una vez que se ha convertido el problema al tipo QUBO, es necesaria hacer una última conversión a un modelo llamado Ising para poder trabajar con los sistemas cuánticos. En este modelo, en lugar de utilizar variables binarias, estos toman los valores 1 o -1, es decir:

$$z_i = 1\ si\ x_i = 0$$
$$z_i = -1\ si\ \ x_i = 1$$

Por lo tanto, se sustituyen los valores x de la expresión anterior:

$$c_{i,j}x_ix_j \rightarrow c_{i,j}\ \frac{1-Z_i}{2}\ \frac{1-Zj}{2}$$

$$dx_i\ \rightarrow d_i\frac{1 - Z_i}{2}$$

Finalmente, se consigue una función que estará representada por variables z. Estas variables, se sustituyen por las matrices correspondientes a la puerta cuántica Pauli Z, y haciendo las operaciones necesarias, se obtiene una matriz final de dimensiones $2^n x 2^n$ denominada Hamiltoniano [46].

Encontrar la solución al problema QUBO que se tenía inicialmente es equivalente a encontrar el valor y el vector propios mínimos de esta última matriz. Teniendo en cuenta la expresión $A\vec{x} = \lambda\vec{x}$, donde A es una matriz y λ es un número, los vectores $\vec{x}$ y los números λ que cumplan esa condición, son denominados el vector y valor propios, respectivamente [47].

Qiskit contiene una clase para este propósito llamado *MinimumEigenOptimizer,* al cual se le indica qué método utilizar para resolver el problema, como los algoritmos cuánticos QAOA, VQE o un algoritmo clásico llamado NumPyMinimumEigenSolver, entre otros. También se encarga de realizar automáticamente las conversiones de los problemas cuadráticos a la representación necesaria, por lo que no es necesario hacerlos manualmente [48].

Estos algoritmos cuánticos como el VQE y QAOA, son algoritmos variacionales híbridos cuánticos-clásicos, en los cuales una parte del algoritmo se ejecuta en un ordenador clásico y otra parte en un ordenador cuántico. La idea general de estos algoritmos como VQE, es crear un circuito inicial (también denominado ansatz) que depende de unos parámetros para crear un estado cuántico. Utilizando el sistema cuántico, se obtiene el valor esperado del estado inicial a través del Hamiltoniano. Este valor se introduce en un algoritmo de optimización en un ordenador clásico y con el resultado se ajustan otra vez los parámetros del circuito inicial, repitiendo así el proceso varias veces [49].

A continuación, se detalla más información sobre uno de los problemas de optimización con los que se puede trabajar, el problema de Corte máximo o Max-Cut.

5.3.1. Max-Cut

Max-Cut es un problema que implica dividir los nodos de un grafo en dos conjuntos, de modo que el número de conexiones o bordes entre los dos conjuntos sea el máximo. Este problema tiene aplicaciones en áreas como la ciencia de redes o física estadística, entre otros [50].

Considerando un grafo de *n* nodos no dirigido G= (V, E) donde |V| = n y los pesos de los nodos son $w_{ij} > 0, w_{ij} = w_{ji}$, $(i,j) \in E$, un corte se define como una partición del conjunto original V en dos subconjuntos.

Teniendo esto en cuenta, la función a maximizar es la suma de los pesos de las conexiones de los nodos que se encuentran en 2 conjuntos distintos:

$$maximize \sum_{i,j} w_{ij} x_i (1 - x_j) + \sum_i w_i x_i$$

En la Figura 5.8 se puede observar un ejemplo del problema Max-Cut con todos los pesos de las conexiones siendo uno. Por un lado, se encuentra el grafo original(arriba), y por otro lado, el mismo problema con una de las soluciones óptimas (abajo). Como contiene 6 nodos y cada uno se puede asignar a 2 conjuntos (azul o rojo), hay $2^6 = 64$ posibles asignaciones, de las cuales se tiene que encontrar una que dé el número máximo de aristas o conexiones entre los 2 conjuntos.

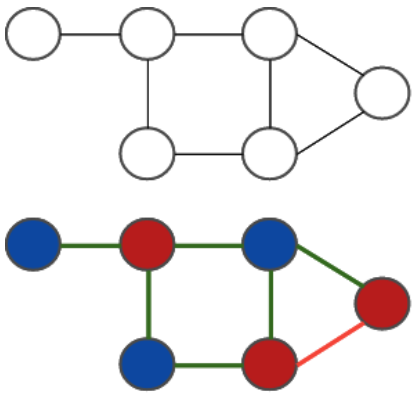


Figura 5.11 - Problema Max-Cut

En este caso, en la solución de abajo las conexiones verdes significan que esa conexión es válida para este problema porque se encuentran entre nodos de conjuntos distintos. Por lo tanto, la solución óptima es 6, ya que una de las conexiones siempre se queda entre nodos del mismo conjunto y no se puede considerar como válido.

Como se puede observar, en este caso se tenían 64 posibles asignaciones, pero a medida que se aumenta el número de nodos, el número de posibles asignaciones que se deben examinar para encontrar la solución óptima también aumenta exponencialmente.

Una vez que se define un grafo, Qiskit dispone de una clase predefinida para este problema, por lo que se puede modelizar fácilmente y generar el problema cuadrático, el cual ya se puede resolver utilizando los métodos que se han mencionado anteriormente.

5.4. Finanzas

Por último, se ha analizado el módulo de las finanzas, un área de la economía que estudia el funcionamiento de los mercados de dinero y capitales, las instituciones que operan en ellos y el valor del dinero en el tiempo y el coste de capital.

Es decir, las finanzas estudian como los agentes económicos (empresas, familias o estado) deben tomar decisiones de inversión, ahorro y gasto en condiciones de incertidumbre. A la hora de tomar las decisiones, los agentes pueden optar por varios tipos de recursos financieros como dinero, bonos, acciones o derivados [51].

Dentro del módulo Qiskit Finance, los problemas financieros se pueden dividir en 2 submódulos; los problemas de optimización y los problemas de estimación. Por un lado, el submódulo de optimización contiene herramientas para optimizar una serie de problemas financieros. Como ejemplo, un problema financiero de optimización es la optimización de cartera (Portfolio Optimization), el cual se analiza más adelante.

Por otro lado, el submódulo de estimación contiene herramientas para trabajar con problemas de valoración de opciones (instrumentos financieros que otorgan al comprador derecho y al vendedor la obligación a realizar una transacción a un precio fijado y en una fecha determinada [52]). Por ejemplo, permite trabajar con el problema de la opción europea, donde el comprador de esta solo podrá ejercerla cuando llegue el vencimiento del contrato y el objetivo es saber si es rentable adquirir esa opción o no [53].

El problema que se va a analizar es el de Portfolio Optimization, el cual tal como indica su nombre, es un problema de optimización y por lo tanto se trabaja con él al igual que otros problemas de optimización que se han visto en la sección anterior. Por lo tanto, una vez definido el problema, se puede modelar el problema cuadrático y resolverlo con los algoritmos ya mencionados.

5.4.1. Portfolio Optimization

El problema de la optimización de cartera es el proceso de seleccionar la mejor cartera (combinación de activos financieros en los que se invierte) del conjunto de todas las carteras consideradas de acuerdo con algún objetivo, generalmente maximizar factores como el rendimiento esperado y minimizar costos como el riesgo financiero [54].

Estas colecciones de activos (títulos o anotaciones contables que otorgan en el comprador derecho a recibir ingreso futuro procedente del vendedor) suelen estar modelados por una distribución de probabilidad, y como se ha dicho, la tarea consiste en crear una cartera con el subconjunto de estos activos que maximice el rendimiento pero que minimicen el riesgo (Ver Figura 5.12).

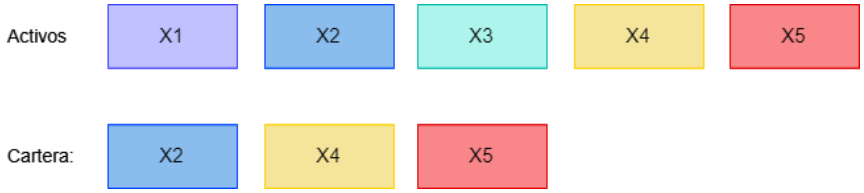


Figura 5.12 - Selección de cartera

La razón de no seleccionar todos los activos es que pueden tener una volatilidad muy alta, es decir, pueden tener una tasa de retorno muy alta o en cambio tener una tasa de retorno muy baja, por lo que en la mayoría de los casos seria mas favorable no escoger este activo. En su lugar, es mejor escoger un activo que tenga una tasa de retorno menor pero que al mismo tiempo tenga una volatilidad menor para asegurarse de que los valores devueltos son más estables.

Cada activo es modelizado por una distribución probabilística multivariada $\vec{X} = (X_1, \dots, X_n)$, $\vec{X} = \mathrm{N}(\mu, \Sigma)$, es decir, está definido por un vector de medias μ que representa el valor de retorno o rendimiento esperado de cada activo y por una matriz de covarianza Σ, que indica como varía un activo y también nos da información sobre el comportamiento de otros activos cuando se utilizan conjuntamente.

Este problema también se puede definir como un problema de minimización que busca la cartera óptima en el que el riesgo sea el mínimo para unos valores esperados. Teniendo esto en cuenta, se puede definir el problema de la siguiente forma:

$$\min_{x \in \{0,1\}^n} q x^T \Sigma x - \mu^T$$
$$subject\ to\ 1^T x = B$$

Donde $x \in \{0,1\}^n$ denota el vector de variables de decisión binarias, el cual indica qué activos escoger ($x[i] = 1$) y cuales no escoger ($x\,[i] = 0$). Por otro lado, el parámetro B indica cuantos de los activos se quieren escoger y el parámetro $q > 0$ es el riesgo que se desea tomar teniendo en cuenta la varianza de los activos, es decir, puede ser tomado como una medida de la volatilidad de toda la cartera.

Por lo tanto, esto en definitiva es un problema cuadrático que se puede resolver con Qiskit siguiendo los mismos procedimientos que con otros problemas de optimización.

6. EXPERIMENTACIÓN

En esta sección se detallan las pruebas que se han realizado para analizar los sistemas cuánticos en las áreas que se han estudiado anteriormente. En todos los casos, se ha medido el tiempo de ejecución de cada problema, los resultados obtenidos y la usabilidad o facilidad con las que es posible implementar cada aplicación.

Antes de empezar con las pruebas, se listan los sistemas en los que se han realizado los experimentos. Entre ellos, se encuentra un ordenador clásico de arquitectura Intel x86, que se ha utilizado tanto para la ejecución de algoritmos clásicos como para simular el entorno cuántico y ejecutar algoritmos cuánticos. Por otro lado, se encuentran los sistemas reales de IBM Q (ibmq_belem, ibmq_lima, ibmq_casablanca) y el simulador cuántico en la nube de IBM Q (ibmq_qasm_simulator).

- **Ordenador clásico/simulador cuántico local:**
  - Procesador: Intel Core i5-7200 de 2,5Ghz
  - Memoria RAM: 12 GB DDR4
- **Ibmq_belem:**
  - Qubits: 5
- **Ibmq_lima:**
  - Qubits: 5
- **Ibmq_casablanca:**
  - Qubits: 7
- **Ibmq_qasm_simulator:**
  - Qubits: 32

6.1. Grover

A continuación, se detallan las pruebas realizadas para probar el algoritmo de Grover, comparando la ejecución de sus circuitos tanto en un simulador local como en un sistema real de IBM para comprobar si se obtienen los resultados esperados.

Siguiendo el cuaderno de Qiskit [37], se prueba la implementación con 2 qubits, utilizando los estados posibles en los que pueden encontrarse los qubits como los elementos a buscar. Por lo tanto, con 2 qubits se tienen los elementos |00⟩, |01⟩, |10⟩ y |11⟩, y entre ellos se quiere encontrar el estado |11⟩. Por lo tanto, una vez inicializados los 2 qubits, se crea el oráculo para marcar el estado |11⟩. En este caso basta con utilizar la puerta CZ (Controlled Z), ya que cambia la amplitud del objetivo en caso de ser 1 cuando el control también es 1.

Después, se implementa el circuito que actúa como difusor para cambiar la amplitud del estado seleccionado, creando de esta manera el circuito final (Ver Figura 6.1).

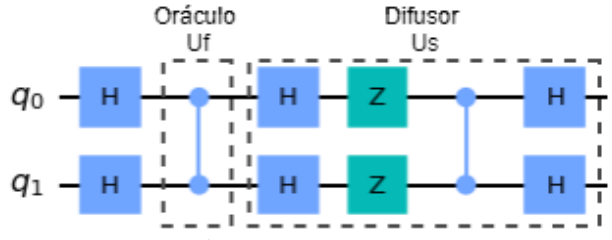


Figura 6.1 - Circuito de Grover para N=2

Por último, se ejecuta el circuito midiendo los resultados. La ejecución se realiza tanto en un simulador como en un sistema cuántico real de IBM, y como se puede ver en la Figura 6.2, el estado con la mayor probabilidad de obtenerse al medir el circuito es el |11⟩. En el caso del simulador, esta probabilidad es del 100%, en cambio, en un sistema real a pesar de no obtener una probabilidad perfecta a causa de errores como el ruido que ocurren en estos sistemas, la probabilidad sigue siendo bastante alta.

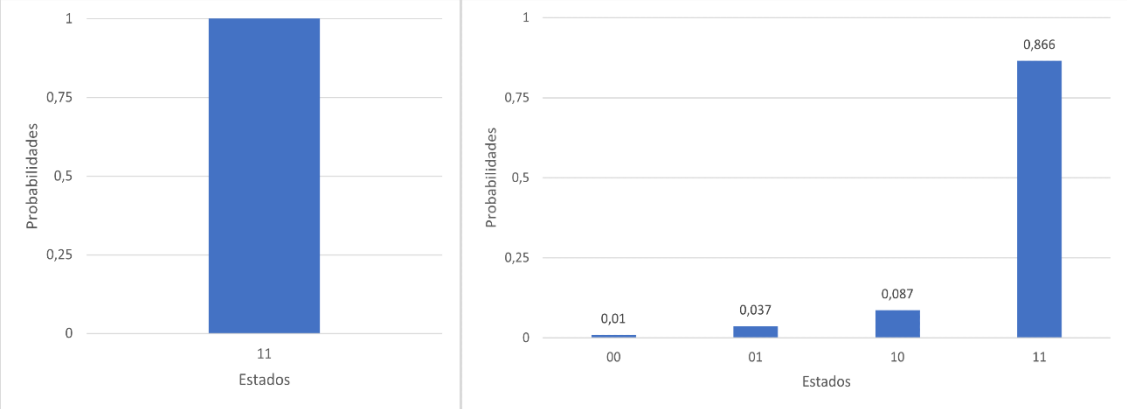


Figura 6.2 - Resultados de ejecución Grover en simulador y en ibmq_belem

También se ha probado a implementar los oráculos para marcar los estados |00⟩, |01⟩ y |10⟩ (ver Figura 6.3). Al ejecutarlos tanto en el simulador como en el sistema real se obtienen los mismos resultados que con la prueba anterior, pero siendo la probabilidad más alta la del estado marcado.

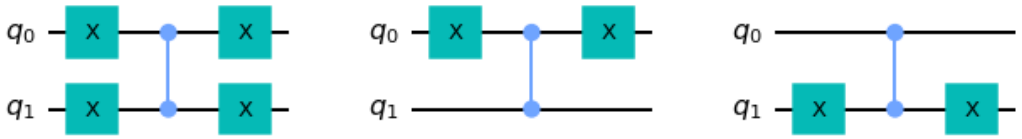


Figura 6.3 - Oráculos para los estados |00⟩, |01⟩ y |10⟩

En estas pruebas solamente ha sido necesario implementar una vez los circuitos del oráculo y el difusor para obtener los resultados deseados, pero como se va a ver a continuación, al aumentar el número de qubits (y por lo tanto el número de estados posibles), es necesario repetir los pasos 2 y 3 para encontrar el estado deseado con una probabilidad alta.

La siguiente prueba que se ha realizado ha sido con 3 qubits, lo cual equivale a tener 8 estados distintos posibles. En este caso, el oráculo es utilizado para marcar el estado |111⟩ (ver Figura 6.4), por lo que se espera que al medir los circuitos sea el que más probabilidad tenga.

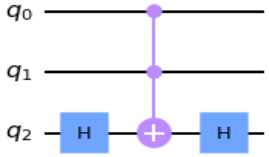


Figura 6.4 - Oráculo para el estado |111⟩

Como se puede observar en el Figura 6.5, son necesarias 2 iteraciones para obtener el estado esperado con la mayor probabilidad, mientras que a partir de ahí la probabilidad empieza a disminuir de nuevo.

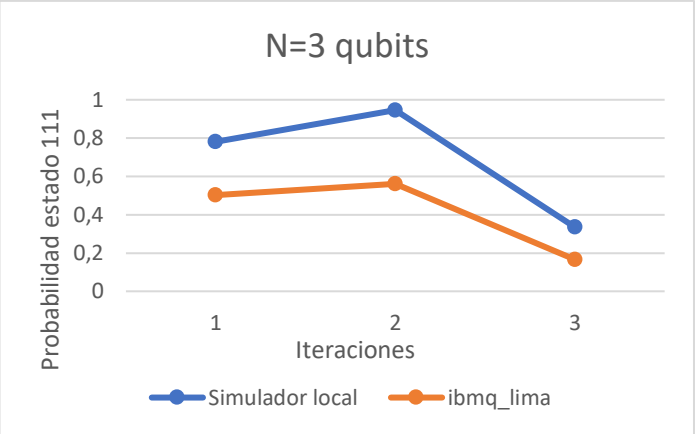


Figura 6.5 - Gráfico de probabilidades al ejecutar Grover con 3 qubits

A pesar de que en el simulador se consigue la probabilidad esperada, en la ejecución en un sistema real de IBM Q (en este caso ibmq_lima porque era el que menos ocupado estaba en ese momento), las probabilidades siguen siendo bajas y repitiendo 2 veces los circuitos del oráculo y el difusor no se consigue la probabilidad esperada.

En la tabla 6.1 se pueden observar los tiempos de ejecución de cada sistema. Por un lado, se muestra el tiempo que se ha necesitado para ejecutar los circuitos, y en el caso de ibmq_lima también se muestra el tiempo necesitado para validarlos. Por otro lado, en la columna total, se ha medido la duración total desde que se lanza el trabajo hasta que se devuelven los resultados utilizando la librería *time* de Python.

Por último, en el caso de ibmq_lima, también se ha añadido una columna para mostrar el tiempo que ha estado en cola antes de ejecutar el trabajo enviado. Como se puede ver, al ser este un sistema público, el tiempo que hay que esperar en la cola de ejecución aumenta considerablemente el tiempo total desde que se lanza el trabajo hasta que se recogen los resultados. Las demás diferencias de tiempo son debidas a los retardos entre enviar y recibir los resultados.

En general, la ejecución en el simulador local es más rápida que en ibmq_lima, no llegando la del simulador ni a un segundo mientras que el otro necesita varios segundos.

Por último, se ha repetido el mismo procedimiento que antes, pero en este caso utilizando 5 qubits y ejecutando el algoritmo con hasta 5 iteraciones. Al igual que con 3 qubits, el simulador va mejorando la probabilidad de obtener el estado $|11111\rangle$ hasta llegar a 4 iteraciones, donde obtiene la mejor probabilidad, mientras que a partir de ahí la probabilidad empieza a empeorar otra vez (ver Figura 6.6).

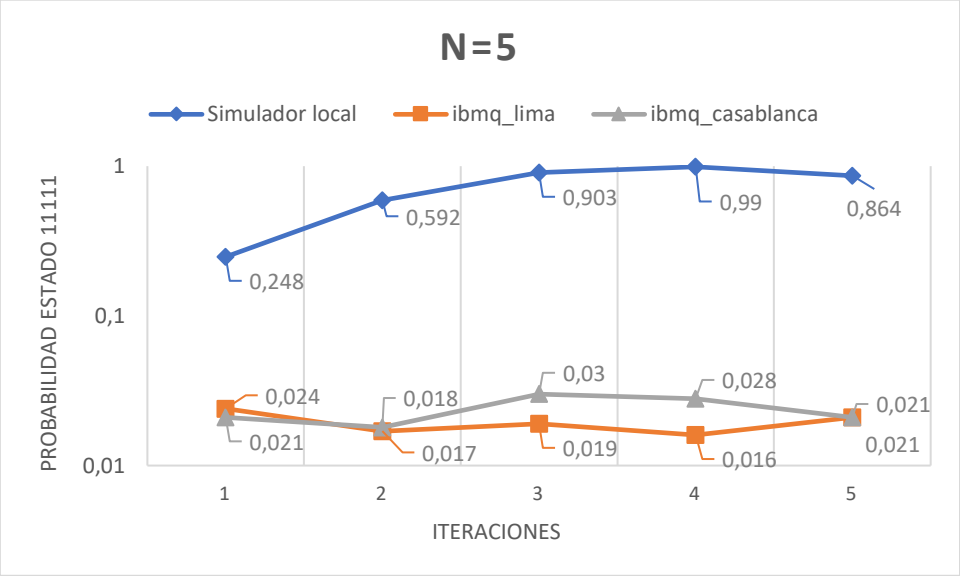


Figura 6.6 - Gráfico de probabilidades al ejecutar Grover con 5 qubits

Sin embargo, lo que teóricamente se espera y se consigue con el simulador, con el sistema real de ibmq_lima sigue sin conseguirse la probabilidad esperada, y en este caso es más evidente ya que en ninguna de las iteraciones la probabilidad de obtener el estado |11111⟩ llega a 0,1. Para asegurarse de estos resultados, también se han repetido las pruebas en el sistema ibmq_casablanca de 7 qubits, y los resultados siguen siendo muy similares.

En cuanto a los tiempos de ejecución, ocurre lo mismo que en el caso anterior, en el que las ejecuciones en los sistemas reales son más lentas que en el simulador local y debido a la cola de espera, pueden llegar a durar varios minutos (ver Tabla 6.2).

En resumen, aunque teóricamente este algoritmo puede ser utilizado para buscar un elemento, esto solo se ha conseguido mediante los simuladores, ya que en los sistemas reales la probabilidad que se obtiene para un estado concreto a partir de los 3 qubits es demasiado baja.

Una de las razones por las que se han podido obtener estos resultados, es debido a la decoherencia que tienen los ordenadores cuánticos actuales. Como se ha visto en la sección 4.2.2, al utilizar puertas controladas, internamente es necesario reubicar los qubits, lo que equivale que se deben utilizar varias puertas auxiliares.

Esto en el caso del oráculo para marcar por ejemplo el estado 111 de 3 qubits significa que se deben utilizar muchas más puertas auxiliares, lo que va incrementando el error, y con 5 qubits este error es aún más grande.

Por lo tanto, con las pruebas realizadas se considera que la ejecución de este algoritmo en los sistemas cuánticos de IBM Q con los que se ha experimentado no es viable, ya que solamente se han obtenido los resultados esperados con los simuladores. Además, aunque no se ha probado, en otros algoritmos que necesiten utilizar este algoritmo como puede ser el caso del algoritmo Quantum Counting [55] se espera también que se obtendrían unos resultados similares.

	Simulador cuántico local	Ibmq_lima			
Iteraciones	**Ejecución**	**Total**	**En cola**	**Validación**	**Ejecución**
1	2.9ms	6m 16s	5m 49s	1.1s	5.046s
2	3.9ms	7m 44s	7m 21s	0.7s	5,28s
3	5ms	23.98s	8.5s	0.93s	4,97s

Tabla 6.1 - Tiempos de ejecución del algoritmo de Grover con 2 qubits

	Simulador cuántico local	Ibmq_lima				Ibmq_casablanca			
Iteraciones	**Ejecución**	**Total**	**En cola**	**Validación**	**Ejecución**	**Total**	**En cola**	**Validación**	**Ejecución**
1	3.9ms	2m 31s	1m 29.7s	1s	5.4s	4m 56s	4m 31s	1.4s	8.7s
2	4ms	9m 12s	8m 49.8s	0.79s	5.6s	28.34s	5.9s	0.95s	9.3s
3	4.9ms	2m 41s	2m 18.3s	0.85s	6.5s	9m 15s	8m 33s	1.1s	10.1s
4	6.1ms	8m 22s	7m 27.8s	1.3s	6.9s	10m 1s	9m 42s	0.8s	9.5s
5	6ms	8m 41s	8m 15.5s	1.3s	7.6s	8m 28s	8m 4.8s	0.93s	11s

Tabla 6.2 - Tiempos de ejecución del algoritmo de Grover con 5 qubits

6.2. Machine Learning

A continuación, se detallan las pruebas que se han realizado para probar el módulo de aprendizaje automático de Qiskit con la implementación del clasificador SVC con un kernel cuántico. Las implementaciones y pruebas se basan en el cuaderno de Qiskit *Quantum Kernel Machine Learning* [42].

En las pruebas se han utilizado dos conjuntos de datos (datasets) distintos. El primero de ellos, es un conjunto de datos llamado Ad-Hoc obtenido mediante la librería *qiskit.ml.dataset* [56]. Esta librería crea una cantidad de puntos aleatoriamente y, como se va a ver en los resultados, está preparada para que se obtengan resultados perfectos utilizando el kernel cuántico.

El segundo dataset es uno obtenido de Kaggle, una plataforma gratuita que dispone de grandes cantidades de conjuntos de datos y con los que permite a los usuarios experimentar y solucionar problemas con temáticas como la ciencia de datos, el análisis predictivo o el aprendizaje automático.

Este dataset contiene datos relacionados con la insuficiencia cardiaca (*Heart Failure Dataset*) [57]. En concreto, contiene diferentes atributos relacionados con el riesgo cardiovascular como el colesterol y la presión sanguínea, y una etiqueta que indica si con estos datos se ha dado un ataque cardiaco o no. El objetivo de entrenar un clasificador con estos datos es predecir si una persona se encuentra en riesgo cardiovascular alto para poder tener un tratamiento temprano.

Para el clasificador SVC, se ha utilizado la librería *sklearn.svm.SVC* de Sckit-learn, una biblioteca para aprendizaje automático de software libre para el lenguaje de programación de Python [58] [59]. Esta librería permite entrenar el clasificador SVC poniendo de parámetro distintos tipos de kernel y obtener una puntuación cuando se utiliza para clasificar otro conjunto de datos.

Para el kernel cuántico, se ha utilizado el objeto *QuantumKernel* de Qiskit y el mapa de características ZZFeatureMap [60]. Se puede observar el circuito cuántico de este mapa de características para 2 qubits en la Figura 6.7.

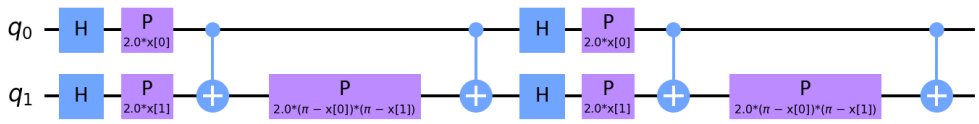


Figura 6.7 - ZZFeatureMap

Con estos experimentos se quiere mencionar que el objetivo no era conseguir entrenar el mejor clasificador (ya que por ejemplo el clasificador SVC de Sckit-learn tiene varios hiperparámetros que pueden hacer variar los resultados y lo óptimo sería encontrar los que mejor se ajusten a un conjunto de datos) si no que el objetivo era probar distintos clasificadores cambiando el tipo de kernel para comprobar si uno cuántico puede ofrecer una mejora respecto a uno clásico.

Para el primer dataset, se disponía de 40 instancias representadas mediante 2 atributos y una clase que podía tomar únicamente 2 valores. De estas 40 instancias, 20 pertenecen a una clase y las otras 20 a la otra clase. Para la evaluación, se disponían de otras 10 instancias nuevas, siendo 5 de una clase y las otras 5 de la otra clase.

En la Figura 6.8 se puede observar el conjunto de datos de entrenamiento, cada punto siendo una instancia representada por 2 valores (eje x, eje y) y siendo de una única clase (azul o naranja).

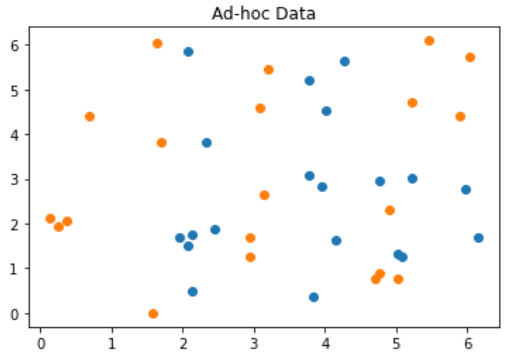


Figura 6.8 - Conjunto de datos Ad-Hoc

Una vez generado el conjunto de datos, se han creado varios clasificadores utilizando el conjunto de entrenamiento para posteriormente evaluarlo utilizando el conjunto de evaluación. Por el lado clásico, se ha entrenado el clasificador SVC varias veces cambiando el parámetro del tipo de kernel; utilizando los valores por defecto, un kernel lineal, polinomial y uno sigmoidal. Por otro lado, se ha entrenado el clasificador SVC, pero utilizando un kernel cuántico y el clasificador QSVC, el cual es un clasificador de IBM Q que extiende de la librería sklearn.svm.SVC por lo que también contiene los métodos de entrenamiento y predicción.

En todos los casos, se ha medido el tiempo de entrenamiento, el tiempo de evaluación y la precisión obtenida al evaluar el conjunto de evaluación (ver tabla 6.3). En el caso del clasificador con los kernel clásicos, todos tienen un tiempo de entrenamiento y evaluación similar, y se ha obtenido la mejor precisión con los parámetros por defecto.

DATASET AD-HOC				
Backend	**Clasificador**	**Tiempo entrenamiento**	**Tiempo evaluación**	**Precisión**
Ordenador clásico	SVC por defecto	9.62ms	1ms	0.7
	SVC kernel lineal	1.78ms	1ms	0.6
	SVC kernel polinomial	5.4ms	0,9ms	0.6
	SVC kernel sigmoidal	1.97ms	0.5ms	0.4
Simulador cuántico local	SVC kernel cuántico	24.06s	10.8s	1
	QSVC	19.04s	10.65s	1
Ibmq_belem	SVC kernel cuántico	5m 24s	3m 18s	1
	QSVC	5m 57s	4m 21s	1
Ibmq_casablanca	SVC kernel cuántico	5m 46s	4m 22s	1
	QSVC	5m 17s	3m 10s	1

Tabla 6.3 - Resultados de los clasificadores en el conjunto de datos Ad-Hoc

En cambio, con el kernel cuántico los tiempos aumentan, siendo la diferencia más considerable en la ejecución en los sistemas reales de ibmq_belm y ibmq_casablanca, pero tal y como se mencionaba al principio, como este conjunto estaba preparado, se obtiene una precisión perfecta.

Cabe destacar que, para las ejecuciones en estos sistemas, Qiskit separa automáticamente los circuitos necesarios en varios trabajos, por lo que no se ha podido obtener el tiempo de ejecución concreto de cada uno. Para ello, al igual que con el algoritmo de Grover, se ha medido el tiempo total utilizando la librería *time* de Python. A este tiempo se le han descartado los tiempos en los que el trabajo estaba en cola, por lo que la información que se encuentra en la tabla es el tiempo total desde el envió de los trabajos hasta recibir los resultados, incluyendo tiempos de validación y ejecución.

Para el segundo conjunto de datos, se disponían de 918 instancias en total, representadas con 11 atributos y su clase. De estas instancias, se han obtenido aleatoriamente 60 para realizar las pruebas y de ellos se ha utilizado el 80% para el entrenamiento y el 20% para la evaluación, quedando 48 y 12 instancias respectivamente.

Después, se ha reducido la dimensionalidad de atributos utilizando la técnica PCA con la librería *sklearn.descomposition.PCA* [61] de Python, consiguiendo los mismos conjuntos, pero representados con 2 y 5 atributos. Esto ha sido necesario para poder hacer las ejecuciones en los sistemas reales, ya que se necesita un qubit por cada variable de entrada. Una vez con estos datos, se han probado los mismos clasificadores que con el otro dataset. En la Figura 6.9 se puede observar el conjunto de datos obtenido al reducir la dimensionalidad a 2.

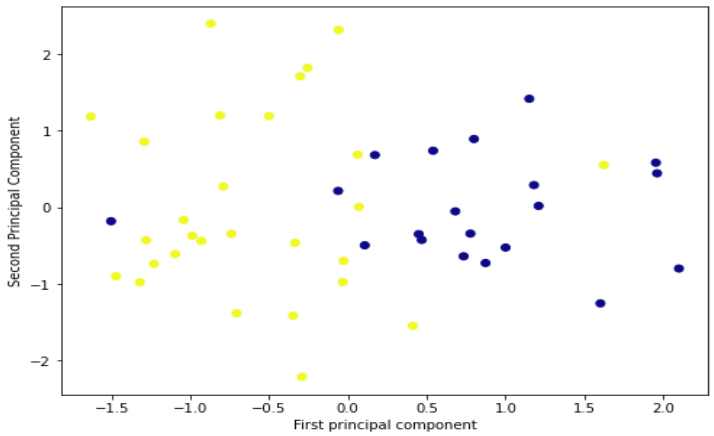


Figura 6.9 - Conjunto de datos Heart Failure reducido a 2 dimensiones

En la tabla 6.4 se muestran los resultados obtenidos tras entrenar los mismos clasificadores que en el caso anterior, pero con el nuevo conjunto de datos que ha sido reducido a 2 dimensiones. A diferencia que con el primer dataset que estaba preparado para obtener mejores resultados con el kernel cuántico, en este caso se puede observar que con los kernel clásicos se obtiene una mejor precisión que con el kernel cuántico y el clasificador QSVC.

Además, ejecutarlos en un simulador en el ordenador local o en el sistema real ibmq_casablanca no ha supuesto una diferencia en los resultados obtenidos en cuanto a la precisión. Sin embargo, sí que se ha notado otra vez que el tiempo de ejecución es bastante superior en este último.

Por último, se ha querido probar si la dimensionalidad inicial afecta en los resultados obtenidos, por lo que se han repetido todas las pruebas, pero con el conjunto inicial reducido a 5 dimensiones con PCA. En los clasificadores con el kernel clásico no se ha visto afectado la precisión obtenida, exceptuando el del kernel polinomial que se ha visto reducido.

En cambio, la precisión con el kernel cuántico y el clasificador QSVC se ha visto reducido y los tiempos tanto para el entrenamiento como para la evaluación han aumentado.

En definitiva, se ha conseguido probar varios clasificadores utilizando kernel clásicos y cuánticos y en el caso de estos últimos, ejecutarlos tanto en un simulador como en un sistema real. En las pruebas realizadas, tanto el entrenamiento como la evaluación con estos kernel ha sido bastante más grande que el de su versión clásica y en el caso del segundo dataset, la precisión obtenida equivaldría a un clasificador aleatorio.

Sin embargo, con el primer conjunto sí que se ha conseguido una precisión perfecta, por lo que puede haber conjuntos de datos que con estos clasificadores obtengan mejores resultados que con los clásicos.

Por último, en cuanto a la usabilidad, se ha visto que las implementaciones que ofrece Qiskit hacen uso de la librería sklearn.svm.SVC, por lo que es posible añadir un nuevo clasificador cambiando únicamente el parámetro del tipo de kernel a utilizar además de indicar el sistema en el que se desea ejecutar.

Por lo tanto, se podría añadir fácilmente un nuevo clasificador a un estudio previo realizado con el clasificador SVC para comprobar si con el kernel cuántico se obtiene un resultado mejor, o directamente utilizar el clasificador QSVC.

DATASET HEART-FAILURE					
Dimensionalidad	**Backend**	**Clasificador**	**Tiempo entrenamiento**	**Tiempo evaluación**	**Precisión**
PCA 2 dimensiones	Ordenador clásico	SVC por defecto	1.98ms	1ms	0.91
		SVC kernel lineal	2ms	1.03ms	0.91
		SVC kernel polinomial	2.96ms	0.98ms	0.83
		SVC kernel sigmoidal	1.1ms	1ms	0.91
	Simulador cuántico local	SVC kernel cuántico	32.11s	16.11s	0.58
		QSVC	37.62s	19.18s	0.58
	Ibmq_casablanca	SVC kernel cuántico	9m 26s	4m 6s	0.58
		QSVC	9m 4s	4m 35s	0.58

Tabla 6.4 - Resultados de los clasificadores en Heart-Failure reducido con PCA a 2dim

DATASET HEART-FAILURE					
Dimensionalidad	**Backend**	**Clasificador**	**Tiempo entrenamiento**	**Tiempo evaluación**	**Precisión**
PCA 5 dimensiones	Ordenador clásico	SVC por defecto	6.88ms	1.56ms	0.91
		SVC kernel lineal	4.95ms	1ms	0.91
		SVC kernel polinomial	2.6ms	0.51ms	0.75
		SVC kernel sigmoidal	5ms	1ms	0.91
	Simulador cuántico local	SVC kernel cuántico	1m 37s	59s	0.5
		QSVC	3m 7s	1m 27s	0.5
	Ibmq_casablanca	SVC kernel cuántico	11m 50s	5m 42s	0.5
		QSVC	15m 52s	8m 46s	0.5

Tabla 6.5 - Resultados de los clasificadores en Heart-Failure reducido con PCA a 5dim

6.3. Optimización: Problema Max-Cut

En esta sección se detallan los experimentos realizados para probar el módulo de optimización de Qiskit para solucionar el problema Max-Cut. Para ello, se han generado 3 grafos distintos, de 4,7 y 20 nodos. En el caso de 4 nodos, solamente se tienen 2 conexiones por nodo, en el caso de 7 nodos, algunos nodos contienen 3 conexiones, y en último caso, todos los nodos tienen 3 conexiones.

Los grafos (ver Figura 6.10) han sido creados utilizando el paquete de Python NetWorkX, el cual permite la creación, manipulación y estudio de la estructura, dinámica y funciones de redes complejas. En definitiva, permite cargar y almacenar redes en formatos de datos estándar y no estándar, generar muchos tipos de redes aleatorias y clásicas, analizar la estructura de la red, construir modelos de red, dibujar redes y mucho más [62].

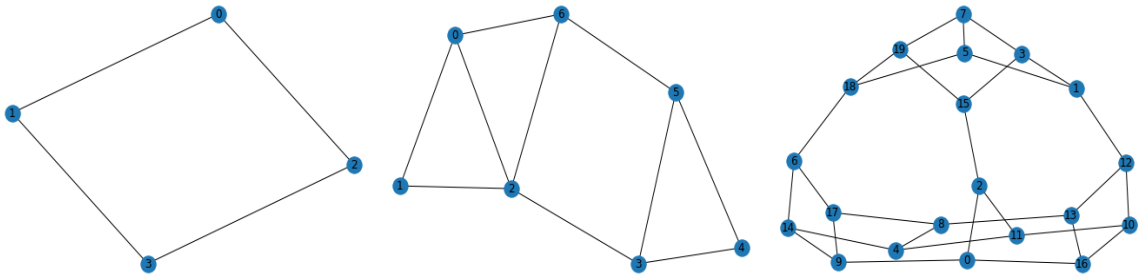


Figura 6.10 - Grafos de 4, 7 y 20 nodos

Siguiendo el cuaderno de Qiskit referente a este problema [50] y utilizando cada grafo generado, se ha utilizado el objeto *Maxcut* de Qiskit para modelizar el problema del grafo y posteriormente con el método to_*quadratic_program()* se ha generado el problema cuadrático a partir de ese modelo.

A continuación, se puede ver el fichero que se genera con el programa cuadrático para el primer grafo utilizando DOcplex, una biblioteca de Python para el modelado de programación matemática [63].

```
\ This file has been generated by DOcplex
\ ENCODING=ISO-8859-1
\Problem name: Max-cut

Maximize
 obj: 2 x_0 + 2 x_1 + 2 x_2 + 2 x_3 + [ - 4 x_0*x_1 - 4 x_0*x_2 - 4 x_1*
x_3 - 4 x_2*x_3]/2
Subject To

Bounds
 0 <= x_0 <= 1
 0 <= x_1 <= 1
 0 <= x_2 <= 1
 0 <= x_3 <= 1
```

Para solucionar cada uno de los problemas, se han utilizado 2 soluciones clásicas y un algoritmo cuántico. En cuanto a la solución clásica, se ha utilizado la fuerza bruta, es decir, probar todas las combinaciones posibles para obtener la solución óptima. Por otro lado, utilizando la clase *MinimumEigenOptimizer,* se han probado el algoritmo clásico *NumPyMinimumEigenSolver* [64] y el algoritmo cuántico *QAOA* [65] para resolver el problema.

En la Figura 6.11 se puede observar el grafo que se obtiene con la solución óptima [0,1,1,0]. Los valores 0 y 1 representan el grupo azul y rojo respectivamente, por lo que, en esta solución, los nodos 0 y 3 pertenecen al grupo azul y los nodos 1 y 2 al rojo. Otra solución óptima seria ese mismo resultado, pero con los grupos invertidos (1,0,0,1), siendo los nodos 0 y 3 del grupo rojo y los nodos 1 y 2 del grupo azul.

Asimismo, en la tabla 6.6, se encuentran los tiempos necesitados para obtener una solución con cada una las formas mencionadas. En este primer caso, probar todas las combinaciones mediante la fuerza bruta ha sido la solución más rápida, pero en los tres casos se ha obtenido la misma solución, con el agrupamiento [0,1,1,0] y el valor óptimo 4.

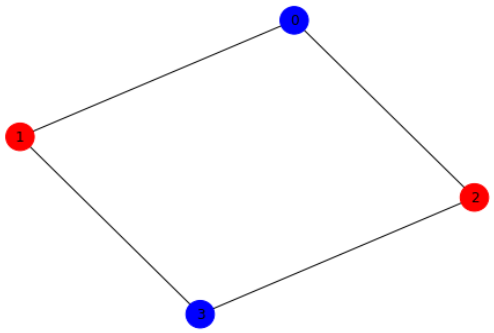


Figura 6.11 - Solución óptima para el grafo de 4 nodos

Una vez obtenido los resultados para el caso de 4 nodos, se han repetido las pruebas para el grafo de 7 nodos después de generar su problema cuadrático. En la Figura 6.12 y en la tabla 6.6 se pueden observar el grafo con una de las soluciones óptimas y los respectivos tiempos necesitados de cada algoritmo.

Al igual que con el anterior grafo, en este caso también la fuerza bruta sigue siendo la solución más rápida y se obtiene la misma solución en los 3 casos, con el agrupamiento [1,0,1,0,0,1,0] y el valor óptimo 8.

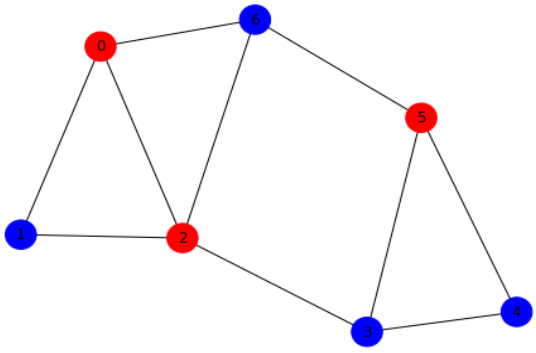


Figura 6.12 - Solución óptima para el grafo de 7 nodos

Por último, siguiendo el mismo procedimiento se ha obtenido la solución y los tiempos de ejecución para el caso del grafo con 20 nodos (ver Figura 6.13 y la tabla 6.6).

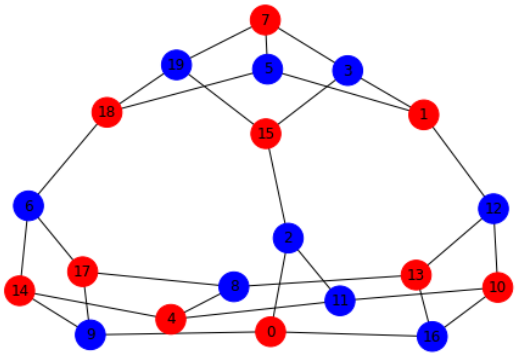


Figura 6.13 - Solución óptima para el grafo de 20 nodos

En este caso, el tiempo para solucionarlo mediante la fuerza bruta ha aumentado considerablemente, siendo el que más ha tardado y obteniendo el mejor tiempo con el algoritmo clásico NumPyMinimumEigenSolver. Con los 2 primeros algoritmos se ha conseguido la agrupación [1,1,0,0,1,0,0,1,0,0,1,0,0,1,1,1,0,1,1,0] y el valor óptimo 28.

		Tiempos totales		
Backend	Algoritmo	N=4	N=7	N=20
Ordenador Clásico	Fuerza Bruta	1.9ms	36ms	10 min 56s
	NumPyMinimumEigenSolver	34ms	58ms	2.86s
Simulador cuántico local	QAOA	48ms	120ms	3.19s

Tabla 6.6 - Tiempos para resolver Max-Cut de 4,7 y 20 nodos

Sin embargo, con el algoritmo QAOA el valor óptimo que se ha obtenido en esa ejecución ha sido 21, con el agrupamiento [0,1,0,1,0,1,1,0,1,1,1,1,0,1,0,0,1,0,0,1]. Por ello, se ha probado a ejecutarlo varias veces más, pero cambiando su parámetro reps (parámetro *p)* que se utiliza para seleccionar la profundidad del circuito inicial en el algoritmo [66].

En la tabla 6.7 se encuentran las soluciones obtenidas junto con el tiempo de ejecución del algoritmo para los distintos valores de reps. Se ha conseguido el valor esperado 28 con el valor de reps 3.

Reps	Agrupamiento	Valor óptimo	Tiempo
1	[0,1,0,1,0,1,1,0,1,1,1,1,0,1,0,0,1,0,0,1]	21	3.17s
2	[0,0,1,1,0,1,1,0,1,1,1,0,1,0,0,0,1,0,0,1]	26	1m 22s
3	[0,0,1,1,1,1,1,0,1,1,0,0,1,0,0,0,1,0,0,1]	28	1m 6s
4	[0,1,1,0,1,0,0,1,0,0,1,0,0,1,1,1,0,1,1,0]	26	52.44s
5	[0,0,0,1,0,1,1,0,1,1,0,1,1,0,0,1,1,0,0,0]	25	42.55s

Tabla 6.7 - Resultados de QAOA para distintos valores de reps

Por último, en las pruebas realizadas con el algoritmo cuántico QAOA, la ejecución se ha realizado solamente en el simulador local, ya que de forma similar a lo que ocurría con el algoritmo de aprendizaje automático, este algoritmo funciona con varias iteraciones, enviando varios trabajos a los sistemas reales. La diferencia es que la cantidad de trabajos que se debe enviar es mucho mayor, y al necesitar el envío de datos entre el sistema local y el sistema cuántico a lo largo de todo el algoritmo, la duración también aumenta considerablemente.

Por lo tanto, después de haber intentado ejecutarlo en un sistema real y tras más de 1 hora de ejecuciones, a causa del reparto de colas equitativas, los tiempos de espera eran demasiado grandes por los que no se ha podido terminar su ejecución.

A raíz de esto, y para poder probarlo realmente en un sistema real, se ha utilizado Qiskit Runtime para crear un único trabajo a ejecutar y obtener los resultados. Este trabajo se ha ejecutado tanto en el simulador ibmq_qasm_simulator de IBM Q como en los sistemas reales ibmq_lima y ibmq_casablanca.

En la Tabla 6.8 se muestra el tiempo de ejecución para el problema de cada grafo en los sistemas ya mencionados, a excepción de los casos en los que no se ha podido ejecutar porque el número de qubits del sistema era inferior a los nodos del grafo.

N	Ibmq_lima	Ibmq_casablanca	ibmq_qasm_simulator
4	46m 28s	1h 1m 12s	31.28s
7	-	1h 1m 53s	1m 20s
20	-	-	22 min 15s

Tabla 6.8 - Tiempos de ejecución de Qiskit Runtime con QAOA para el problema Max-Cut

Para el problema de 4 y 7 nodos, se ha obtenido la solución correcta en todas las ejecuciones, con el valor óptimo 4 y 8 respectivamente. Sin embargo, al igual que pasaba con el simulador local, al resolver el problema de 20 grafos con el valor de reps 1, en la ejecución con Runtime tampoco se obtiene la solución óptima. Por eso se ha intentado ejecutarlo cambiado el valor de reps, pero el programa daba un error por lo que no se ha podido ejecutar con valores de reps distintos a 1.

Como conclusión de esta aplicación y el módulo de optimización, en cuanto a la usabilidad permite modelizar de una forma sencilla distintos tipos de problemas o cargarlos de ficheros ya existentes para poder solucionarlos con algoritmos cuánticos.

En cambio, al igual que ocurría con el módulo de Machine Learning, los tiempos de ejecución que se han conseguido en los sistemas reales son más altos que los obtenidos con un simulador en el ordenador local. Además, aunque con Qiskit Runtime se esperaba resolver el problema de los tiempos y de cierta manera se ha conseguido ya que ha permitido terminar la ejecución, siguen siendo muy altos. La mayor diferencia se puede apreciar al ejecutar el mismo programa de Runtime en los sistemas reales y en el simulador ibmq_qasm_simulator de IBM Q.

Adicionalmente, no se ha conseguido encontrar la razón por la que no se ha podido ejecutar este programa con un valor distinto del parámetro reps a 1.

6.4. Finanzas: Optimización de cartera

En esta sección se detallan las pruebas realizadas con el módulo Qiskit Finance para resolver el problema de optimización de cartera, basándose en el cuaderno de Qiskit [67]. Las pruebas se han realizado con 2 conjuntos de datos, uno con datos generados aleatoriamente y el otro con datos reales obtenidos de Quandl, una plataforma que proporciona conjuntos de datos económicos y financieros [68].

Para generar el primer conjunto de datos se ha utilizado la clase *RandomDataProvider* de Qiskit, indicando como parámetro la cantidad de activos a generar en un rango de tiempo. Para este primer conjunto, se han utilizado 4 activos con datos en un rango de 30 días (ver Figura 6.14).

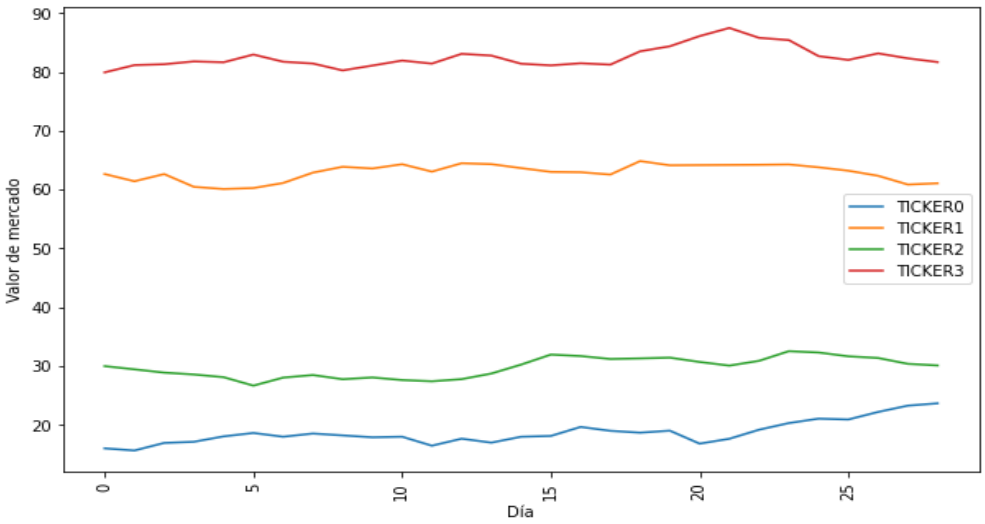


Figura 6.14 - Conjunto de datos aleatorio para optimización de cartera

Una vez generados los datos, utilizando los métodos del objeto *RandomDataProvider*, se obtienen los valores de retorno esperados: $[0.01528439, -0.00078095, 0.00051792, 0.00087001]$).

Posteriormente se crea un objeto de tipo *PortfolioOptimization* indicando como parámetro el riesgo que se desea tomar y cuantos de los activos hay que seleccionar, en este caso 0.5 y 2 respectivamente.

A partir de esa clase, se puede utilizar el método to_quadratic_problem() para modelizar el problema a resolver.

Para resolver este problema, se ha utilizado otra vez la clase *MinimumEigenOptimizer,* y se ha probado a resolver el problema con el algoritmo clásico *NumPyMinimumEigenSolver* y el algoritmo cuántico *VQE* [69].

Al igual que en el problema de optimización, se puede observar el fichero que se genera con el problema modelizado para este primer conjunto:

```
\ This file has been generated by DOcplex
\ ENCODING=ISO-8859-1
\Problem name: Portfolio optimization

Minimize
 obj: - 0.015284386652 x_0 + 0.000780952145 x_1 - 0.000517920547 x_2
      - 0.000870005837 x_3 + [ 0.002541388592 x_0^2 + 0.000146804433 x_0*x_1
      + 0.000257201062 x_0*x_2 - 0.000199722426 x_0*x_3 + 0.000258486713 x_1^2
      + 0.000106085519 x_1*x_2 + 0.000088963242 x_1*x_3 + 0.000791504681 x_2^2
      - 0.000247774763 x_2*x_3 + 0.000197892585 x_3^2 ]/2
Subject To
 c0: x_0 + x_1 + x_2 + x_3 = 2

Bounds
 0 <= x_0 <= 1
 0 <= x_1 <= 1
 0 <= x_2 <= 1
 0 <= x_3 <= 1

Binaries
 x_0 x_1 x_2 x_3
End
```

Sin embargo, tal y como pasaba con QAOA, la ejecución en los sistemas reales no se ha podido completar por la larga duración que tienen y las colas que se generan. Por lo tanto, se ha intentado ejecutarlo con Runtime en un único trabajo. Esta ejecución se ha realizado tanto en ibmq_casablanca como en el simulador ibmq_qasm_simulator.

En la tabla 6.9 se encuentran los resultados obtenidos. En todas las ejecuciones el valor óptimo que se ha conseguido ha sido -0.0149 con la selección óptima [1,0,0,1]. Por lo tanto, la mejor selección de los activos es el 1 y el 4, y solamente se han seleccionado 2 porque es lo que se ha especificado como parámetro en la creación del problema.

Las ejecuciones en local, tanto con el algoritmo clásico como con los cuánticos se han ejecutado en un tiempo similar sin llegar a 1 segundo. Sin embargo, al igual que en pruebas anteriores, los tiempos de las ejecuciones con Runtime aumentan considerablemente, siendo el mayor tiempo la del algoritmo VQE cuando se ejecuta en el sistema real ibmq_casablanca, llegando hasta casi las 4 horas.

RANDOM DATA PROVIDER		
Backend	Algoritmo	Tiempo
Ordenador clásico	NumPyMinimumEigenSolver	0.16s
Simulador cuántico local	VQE	0.28s
Ibmq_qasm_simulator	VQEProgram	44.7s
Ibmq_casablanca	VQEProgram	3h 38m

Tabla 6.9 - Tiempos obtenidos en problema Portfolio Optimization con el primer conjunto

Para obtener el segundo conjunto de datos, se ha utilizado el objeto *WikipediaDataProvider*, que permite obtener datos reales de Quandl. Para ello, es necesario indicar un rango de tiempo junto con los tickers (códigos bursátiles que se utilizan para identificar de forma abreviada las acciones de una determinada empresa).

Para obtener estos datos, se ha utilizado la versión gratuita, por lo que solo ha sido posible acceder a los precios de cierre históricos de fechas anteriores y por ello se ha seleccionado el primer mes de 2016.

Las empresas de las que se ha obtenido la información indicando sus tickers tenían que ser de empresas estadounidenses, ya que Quandl es parte de NASDQ, el segundo mercado de valores y bolsa de valores automatizada y electrónica más grande de Estados Unidos [70].

Los Tickers de las empresas seleccionadas han sido las siguientes: Amazon (AMZN), Apple (AAPL), Tesla (TSLA), Facebook (FB), NVIDIA (NVDA), Microsoft (MSFT) y Alphabet (GOOGL) y se puede observar los datos obtenidos en la Figura 6.15, siendo el valor de retorno esperado de cada uno el siguiente: $\mu = [-0.00400, -0.0038, -0.00843, 0.00606, -0.00522, 0.00057, 0.00033])$.

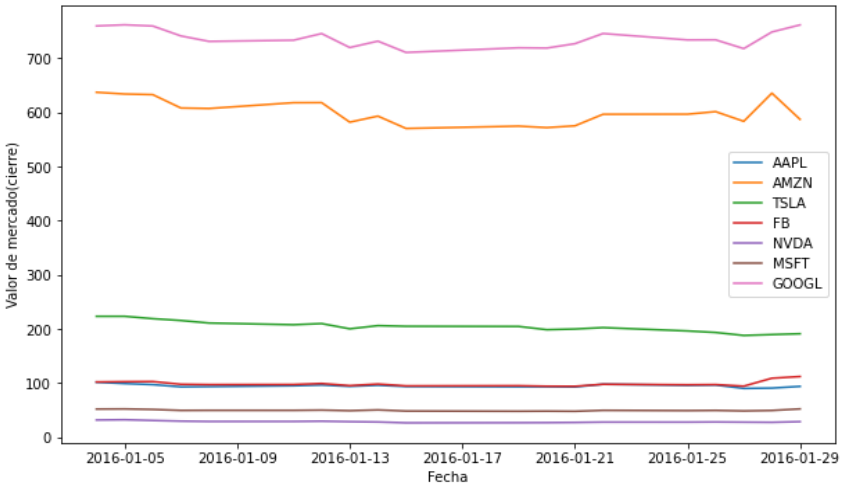


Figura 6.15 - Conjunto de datos obtenido de Quandl para optimización de cartera

Una vez obtenido el conjunto de datos, se ha seguido el mismo procedimiento que con el conjunto anterior para crear el problema cuadrático con el mismo factor de riesgo y con la penalización de tener que seleccionar 4 de los activos. Posteriormente se ha resuelto con los mismos algoritmos, cuyos tiempos se pueden ver en la Tabla 6.10 y en todos los casos el valor obtenido ha sido 0.0024 con la selección óptima [1,0,0,1,0,1,1].

WikipediaDataProvider		
Backend	Algoritmo	Tiempo
Ordenador clásico	NumPyMinimumEigenSolver	0.059s
Simulador cuántico local	VQE	1.39s
Ibmq_qasm_simulator	VQEProgram	2m 31s
Ibmq_casablanca	VQEProgram	-

Tabla 6.10 - Tiempos obtenidos en problema Portfolio Optimization con el segundo conjunto

La ejecución de Runtime en el sistema ibmq_casablanca no se ha podido completar ya que después de alrededor de 5 horas de ejecución ha devuelto un error por exceder el tiempo límite de ejecución en el sistema (ver Figura 6.16).

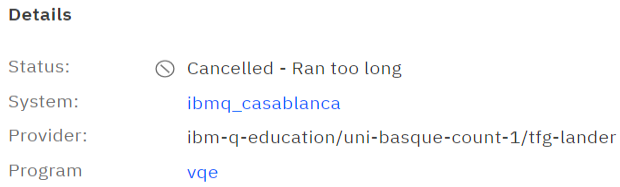


Figura 6.16 - Error obtenido con Qiskit Runtime VQE

Como conclusión, en cuanto a la usabilidad, al igual que los otros módulos permite generar o cargar distintos conjuntos de datos financieros y resolverlos utilizando algoritmos clásicos o cuánticos.

Sin embargo, sigue ocurriendo lo mismo con los tiempos de ejecución en los sistemas reales, y en el caso del algoritmo VQE no se ha podido completar la ejecución con Runtime en el segundo conjunto a causa de exceder el tiempo límite de ejecución.

6.5. Resumen de resultados

Para finalizar con este apartado de experimentación y una vez analizados los resultados obtenidos en cada aplicación, el objetivo de esta sección es hacer un resumen y comparativa general de todos los resultados obtenidos en conjunto.

En primer lugar, se ha visto que, en cuanto a la usabilidad, IBM Q dispone de varias herramientas que lo favorecen, ya que permiten cargar o generar distintos conjuntos de datos para poder trabajar con ellos utilizando tanto algoritmos clásicos como algoritmos cuánticos.

Además, en los casos en los que se utilizan algoritmos o soluciones cuánticas, permite seleccionar de una forma sencilla el backend o el dispositivo en el que se quiere ejecutar, ya sea en un simulador utilizando el ordenador local o en uno de los sistemas de IBM Q. En el caso del módulo del aprendizaje automático, también se ha visto que el clasificador se utiliza junto con una librería clásica, por lo que cambiando únicamente un parámetro, es posible añadir un nuevo clasificador que utilice una parte cuántica a un estudio previo realizado con clasificadores que utilizasen solamente una solución clásica.

Por otro lado, en cuanto a los resultados obtenidos, utilizando algoritmos cuánticos en los problemas de Max-Cut y Portfolio Optimization se han obtenido los mismos resultados que con los algoritmos clásicos, a excepción de un caso con el algoritmo QAOA en el problema Max-Cut, el cual se ha tenido que repetir varias veces cambiando uno de sus parámetros para obtener el valor esperado.

En el problema del aprendizaje automático, a pesar de que con uno de los conjuntos se han obtenido peores resultados utilizando la solución cuántica, sí que se ha visto que con el otro conjunto ocurría lo contrario, es decir, se obtenían mejores resultados con la solución cuántica comparadas con las obtenidas en la solución clásica.

En los casos mencionados hasta ahora, la ejecución de los algoritmos cuánticos en un sistema real no ha supuesto una diferencia en los resultados obtenidos con la ejecución en un simulador

local. Sin embargo, con las pruebas del Algoritmo de Grover, los resultados obtenidos al ejecutarlos en los sistemas de IBM Q no correspondían con los del simulador.

Para confirmar estos resultados, se ha encontrado otro estudio realizado con el algoritmo de Grover en los sistemas cuánticos de IBM [71]. En este estudio, se ha probado la ejecución del algoritmo tanto en un simulador local como en sistemas reales de IBM Q y se han obtenido unos resultados similares a los de este trabajo, en los que las probabilidades obtenidas en los sistemas reales son inferiores a los que se consiguen en el simulador y no llegan al valor esperado.

En cuanto a los tiempos de ejecución, se ha visto en todos los casos que los tiempos en los sistemas de IBM Q han sido mayores que en las del entorno simulado en el ordenador local. Para el algoritmo de Grover, la diferencia de tiempos no ha sido muy significativa ya que solamente era necesaria la ejecución de un único circuito.

Sin embargo, con el módulo del aprendizaje automático era necesario ejecutar varios circuitos para entrenar los clasificadores con el kernel cuántico y los tiempos tanto de entrenamiento como de predicciones han supuesto una diferencia más grande al ser ejecutados en los sistemas reales.

Adicionalmente, se ha encontrado otro trabajo que realiza un estudio sobre la comparación del rendimiento entre algoritmos de aprendizaje automático de IBM Q [72]. En este estudio se han obtenido también unos resultados similares a los de este trabajo, con unos tiempos de ejecución de los algoritmos de aprendizaje automático superiores al ser ejecutados en sistemas reales.

Para los algoritmos QAOA y VQE en los problemas Max-Cut y Portfolio Optimization el número de circuitos que se tenían que ejecutar era aún mayor, y por las colas de espera que se generaban no se ha podido terminar su ejecución en estos sistemas.

Por ello, también se ha probado a utilizar la funcionalidad Qiskit Runtime para enviar un único trabajo y poder solucionar ese problema. A pesar de que se ha conseguido terminar la ejecución en los sistemas que lo han permitido, los tiempos que se han obtenido siguen siendo muy grandes.

La mayor diferencia se ha observado cuando el mismo programa de Qiskit Runtime ha sido ejecutado en uno de los simuladores cuánticos en la nube de IBM Q y en sus sistemas reales, necesitando en estos últimos hasta varias horas para terminar la ejecución mientras que en el simulador se completaba en varios minutos o incluso en segundos.

Además, con el programa de Qiskit Runtime con el algoritmo QAOA con el último grafo del problema Max-Cut no se ha podido obtener la solución esperada ya que al cambiar uno de sus parámetros devolvía un error. En el caso del programa con VQE para el segundo conjunto de datos en el problema de Portfolio Optimization la duración excedía el tiempo límite y tampoco se ha podido terminar su ejecución.

Por último, se ha encontrado otro estudio realizado sobre las métricas que se pueden utilizar para medir el rendimiento de computadoras cuánticas [73]. En este estudio se concluye que la mayoría del tiempo de ejecución de los sistemas cuánticos actuales es debido a los tiempos utilizados en compilar y transferir los datos, por lo que realmente las ejecuciones serían más rápidas.

7. CONCLUSIONES Y TRABAJO FUTURO

En esta última sección, se hace un análisis global de todo el proyecto, observando si se han cumplido los objetivos, los problemas o retrasos que ha habido en los tiempos de cada tarea, y también se menciona como se podría expandir este proyecto o los trabajos futuros que se podrían realizar.

7.1. Tiempo dedicado

Al principio del proyecto, se había hecho una estimación de los tiempos necesarios para completar cada tarea. Sin embargo, a la hora de realizarlos, estos tiempos han ido variando, necesitando en algunas ocasiones menos tiempo de lo esperado, y en otras ocasiones superando el tiempo estimado. Por ello, se ha hecho una tabla comparando el tiempo estimado y el tiempo real que se ha necesitado en cada tarea (Ver Tabla 7.1).

Tareas	Estimado	Real
Gestión	**30**	**21**
Reuniones	15	9
Objetivos	4	4
Estimación de tiempos	3	3
Diagrama de Gantt	8	5
Análisis	**110**	**124**
Estudio Computación Cuántica	35	32
Estudio Aplicaciones	40	52
Revisión Sistemas actuales	15	15
Selección y estudio de sistema	20	25
Implementación y experimentación	**80**	**89**
Algoritmo cuántico	20	15
Aprendizaje automático	20	32
Optimización	20	24
Finanzas	20	18
Documentación	**100**	**110**
Búsqueda material de apoyo	20	15
Redactar la memoria	70	85
Preparar la defensa	10	10
TOTAL	**320**	**344**

Tabla 7.1 - Tiempos estimados y reales de cada tarea

En general, en algunas tareas se ha necesitado menos tiempo mientras que en otras se ha excedido el tiempo estimado y el tiempo total del proyecto ha aumentado un poco pero tampoco ha sido una diferencia demasiado grande.

Una de las mayores diferencias se puede observar en la tarea de buscar información sobre las aplicaciones de la computación cuántica. En un principio, a parte de las aplicaciones que se han ido mencionando a lo largo del proyecto, también se buscó información sobre otras aplicaciones como generadores de números aleatorios, redes neuronales cuánticas u otros algoritmos cuánticos como el algoritmo de Shor que puede ser utilizado para descomponer un número en factores.

Sin embargo, al ver que se estaba sobrepasando el tiempo estimado y esto conllevaría también mucho más tiempo a la hora de tener que experimentar con ellos, se decidió descartarlos y centrarse en una única aplicación en cada área, por lo que en la memoria final tampoco se han mencionado.

Otra de las tareas en las que se ha invertido más tiempo de lo esperado ha sido la experimentación con el módulo de aprendizaje automático de Qiskit. En este caso, además de las pruebas relacionadas con la computación cuántica, ha sido necesario aprender y probar a utilizar la librería Sckit-learn para la carga y preprocesado de datos y posteriormente el uso de los clasificadores.

Adicionalmente, al ser uno de los primeros módulos de Qiskit con el que se experimentaba, en un principio los resultados que se estaban obteniendo respecto a los tiempos de ejecución no eran los que se esperaban, por lo que se han repetido varias veces las pruebas para asegurarse de estos resultados.

7.2. Trabajo futuro

En cuanto al trabajo futuro, este proyecto es posible extenderlo de varias formas. Por un lado, tal y como se ha ido mencionando, dentro de cada módulo analizado, solamente se ha probado con algunas de las aplicaciones, como el algoritmo SVM con un kernel cuántico, o los problemas Max-Cut y Portfolio Optimization. Por ello, un trabajo futuro sería analizar algunas de las otras aplicaciones que permiten trabajar los módulos de Qiskit, como las redes neuronales cuánticas o incluso el módulo Qiskit Nature para problemas relacionados con la naturaleza que no se ha llegado a probar.

Asimismo, también es posible centrarse más en las aplicaciones ya mencionadas para intentar conseguir unos mejores resultados, ya sea probando a cambiar los parámetros de los que disponen o utilizando algoritmos similares. Como ejemplo, en el módulo de aprendizaje automático se podría probar a cambiar el mapa de características o modificar sus valores para el kernel cuántico.

Otra de las áreas interesantes en las que también se podría probar la computación cuántica es en la criptografía, con problemas como la generación de números aleatorios o en algoritmos de encriptación o desencriptación.

Una de las limitaciones del trabajo ha sido la cantidad de qubits de los sistemas reales que se podía utilizar con IBM Q, con un máximo de 7 qubits. Como se ha visto en las pruebas, para estas cantidades pequeñas las ejecuciones se podían realizar con algoritmos clásicos y además obtenían tiempos más rápidos. También se ha podido simular el entorno cuántico en el

ordenador local, por lo que las ejecuciones en este último también han sido más rápidas que en los sistemas reales ya que no sufre de retardos ocasionados por el envío y recepción de datos o la preparación del entorno.

Por lo tanto, ya que en IBM Q no se pueden utilizar sistemas con más cantidad de qubits si no se pertenece a IBM Q Network siendo por ejemplo parte de una empresa, un trabajo futuro sería volver a realizar las pruebas pero expandiéndolas a sistemas más grandes, como mínimo de 50 qubits, para comprobar si el ordenador local ya no podría simular el entorno y los algoritmos clásicos tampoco podrían resolver los problemas en un tiempo razonable. Para ello, se podrían utilizar otros de los sistemas mencionados en este trabajo, como Amazon Braket o Azure Quantum.

Por otro lado, para comparar los sistemas cuánticos, en este trabajo solamente se ha tenido en cuenta los resultados obtenidos y los tiempos de ejecución medidos con la librería *time* de Python o las obtenidas con Qiskit en los casos que ha sido posible. Sin embargo, recientemente IBM ha propuesto una métrica de rendimiento llamado CLOPS (CIrcuit Layer Operation Per Second) para sus sistemas cuánticos [74].

CLOPS es una métrica relacionada a la cantidad de circuitos cuánticos que un sistema cuántico puede ejecutar por unidad de tiempo. Esto no se refiere únicamente a la velocidad a la que procesa y completa con éxito la carga de trabajo, sino que también tiene en cuenta la latencia ocasionada por la interacción entre el sistema cuántico y el ordenador o sistema clásico.

Esta métrica se puede considerar como el análogo a la métrica FLOPS (Floating point Operation Per Second) que se utiliza en la computación clásica para medir el número de operaciones flotantes por segundo de un sistema [75] [76]. Por lo tanto, otro trabajo futuro podría ser el análisis o comparativa de los sistemas cuánticos utilizando esta métrica para cuantificar el rendimiento de los algoritmos.

7.3. Conclusiones generales

En cuanto a los objetivos principales, se ha conseguido aprender los principios básicos de la computación cuántica y algunas de las aplicaciones en las que se puede utilizar. También se han observado algunas de los sistemas que ofrecen las empresas para trabajar con sistemas cuánticos y tras seleccionar IBM Q como el sistema a utilizar, se han realizado varios experimentos con él, probando un algoritmo cuántico y aplicaciones en las áreas del aprendizaje automático, optimización y las finanzas.

Los resultados generales ya han sido discutidos en la sección anterior, sin embargo, se quiere mencionar que, debido a la baja disponibilidad de estos sistemas, si no se creaba una reserva, en ocasiones había unas colas de trabajo demasiado largas, imposibilitando en ese momento ejecutar aplicaciones o circuitos.

Por otro lado, en cuanto al sistema IBM Q y Qiskit se ha visto que están muy bien documentados tanto a nivel de teoría general de la computación cuántica como a nivel de aplicaciones en las que se puede utilizar.

Sin embargo, uno de los problemas que se han visto es que al seguir Qiskit actualizándose y migrando algunas de sus funcionalidades como el módulo Qiskit Aqua que ha dejado de existir, a la hora de realizar el trabajo todavía se mantenían las documentaciones de varias versiones, por lo que en ocasiones ha llegado a generar confusión ya que se estaba observando una

documentación antigua que referenciaba ejemplos o tutoriales que ya no existían o habían cambiado de nombre.

Con Qiskit también se ha visto que permite generar códigos a nivel de circuitos o utilizar clases más generales para solucionar problemas y tiene facilidad para generar o cargar conjuntos de datos con los que trabajar.

Con todo, se ha conseguido adentrar en el mundo de la computación cuántica y tener una visión más real de las características o las funcionalidades que ofrecen, probando distintas aplicaciones y aprendiendo las bases de esta tecnología. Aunque con las pruebas realizadas no se considera que estos sistemas puedan sustituir los ordenadores que conocemos actualmente, sí que podrían ser de gran utilidad en un futuro, aunque como se ha dicho, también habría que probarlos en sistemas más grandes para comprobar si pueden resolver actualmente problemas que en un simulador o en un ordenador clásico no se podrían.

8. BIBLIOGRAFÍA

[1] «Aplicaciones Inteligencia Artificial»

https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7640807/

[2] «Beneficios computación cuántica» https://www.futurelearn.com/info/blog/what-is-quantum-computing

[3] «Computación Cuántica»

https://es.wikipedia.org/wiki/Computaci%C3%B3n_cu%C3%A1ntica.

[4] «Compañias computación cuántica» https://revistabyte.es/actualidad-it/computacion-cuantica-que-empresas-lideran-el-sector/

[5] «Sueldo medio Ingeniero» https://www.jobted.es/salario/ingeniero-inform%C3%A1tico

[6] «Sueldo Neto» https://cincodias.elpais.com/herramientas/calculadora-sueldo-neto/#tabla_resultados

[7] «Cotización Seguridad Social» https://www.infoautonomos.com/blog/cuanto-cuesta-contratar-un-trabajador/

[8] «Sueldo medio profesorado» https://www.jobted.es/salario/profesor-universidad

[9] «Learn quantum computing: a field guide»

https://quantum-computing.ibm.com/composer/docs/iqx/guide/

[10] «Learn Quantum Computation using Qiskit» https://qiskit.org/textbook/what-is-quantum.html

[11] «Qubit» https://es.wikipedia.org/wiki/Cúbit

[12] «Producto de Kronecker» https://es.wikipedia.org/wiki/Producto_de_Kronecker

[13] «Qiskit Single Qubit Gates» https://qiskit.org/textbook/ch-states/single-qubit-gates.html#2.-Digression:-The-X,-Y-&-Z-Bases-

[14] «Decoherencia Cuántica» https://blogs.scientificamerican.com/observations/the-problem-with-quantum-computers/

[15] «IBM Q, Qubit» https://quantum-computing.ibm.com/composer/docs/iqx/guide/the-qubit

[16] «IBM Quantum» https://www.ibm.com/quantum-computing/

[17] «IBM Q Network» https://www.ibm.com/quantum-computing/network/members/

[18] «Qiskit» https://qiskit.org/documentation/

[19] «Amazon Braket» https://docs.aws.amazon.com/braket/latest/developerguide/what-is-braket.html

[20] «Precios Amazon Braket» https://aws.amazon.com/es/braket/pricing/

[21] «Azure Quantum» https://azure.microsoft.com/es-es/resources/development-kit/quantum-computing/#overview

[22] «Proveedores Azure» https://docs.microsoft.com/es-es/azure/quantum/qc-target-list

[23] «PennyLane» https://pennylane.readthedocs.io/en/stable/

[24] «Lista Sistemas actuales» https://en.wikipedia.org/wiki/Quantum_programming

[25] «IBM Q Proveedores»
https://quantum-computing.ibm.com/lab/docs/iql/manage/provider/

[26] «IBM Q Servicios» https://quantum-computing.ibm.com/services?services=systems

[27] «IBM Q Colas de trabajo»
https://quantum-computing.ibm.com/services/docs/services/manage/systems/queue/

[28] «IBM Q Composer» https://quantum-computing.ibm.com/composer/docs/iqx/

[29] «IBM Q Lab» https://quantum-computing.ibm.com/lab/docs/iql/

[30] «Elementos Qiskit» https://qiskit.org/documentation/stable/0.24/the_elements.html

[31] «Ignis» https://github.com/Qiskit/Qiskit-ignis

[32] «Migración Qiskit Aqua» https://github.com/Qiskit/qiskit-aqua

[33] «Transpilador» https://qiskit.org/documentation/apidoc/transpiler.html

[34] «Qiskit Runtime» https://quantum-computing.ibm.com/lab/docs/iql/runtime/

[35] «Arquitectura Qiskit Runtime»
https://iqx-docs.quantum-computing.ibm.com/_images/Qiskit_Runtime_architecture1.png

[36] «Blog IBM Q Runtime» https://research.ibm.com/blog/120x-quantum-speedup

[37] «Algoritmo de Grover» https://qiskit.org/textbook/ch-algorithms/grover.html

[38] E. Alpaydın, Introduction To Machine Learning, 2010.

[39] «Machine Learning»
https://www.sas.com/es_es/insights/analytics/machine-learning.html

[40] «PCA» https://www.cienciadedatos.net/documentos/py19-pca-python.html

[41] «SVM» https://la.mathworks.com/discovery/support-vector-machine.html

[42] «Qiskit QSVM»
https://github.com/Qiskit/qiskit-machine-learning/blob/main/docs/tutorials/03_quantum_kernel.ipynb

[43] «Supervised learning with quantum enhanced feature spaces» https://arxiv.org/pdf/1804.11326.pdf

[44] «Quadratic Programs»

https://qiskit.org/documentation/optimization/tutorials/01_quadratic_program.html

[45] «Lista problemas optimización»

https://qiskit.org/documentation/optimization/tutorials/09_application_classes.html

[46] «Variational Quantum Optimization» https://arxiv.org/pdf/1907.04769.pdf

[47] «Eigenvectors and Eigenvalues»

https://qiskit.org/textbook/ch-appendix/linear_algebra.html#Eigenvectors-and-Eigenvalues

[48] «MinimumEigenOptimizer» https://qiskit.org/documentation/optimization/stubs/qiskit_optimization.algorithms.MinimumEigenOptimizer.html

[49] «A variational eigenvalue solver on a quantum processor»

https://arxiv.org/pdf/1304.3061.pdf

[50] «Qiskit Optimization Max-Cut» https://github.com/Qiskit/qiskit-optimization/blob/stable/0.3/docs/tutorials/06_examples_max_cut_and_tsp.ipynb

[51] «Finanzas» https://es.wikipedia.org/wiki/Finanzas

[52] «Opción financiera» https://es.wikipedia.org/wiki/Opci%C3%B3n_financiera

[53] «Opción Europea» https://economipedia.com/definiciones/opcion-europea.html

[54] «Portfolio Optimization»

https://es.wikipedia.org/wiki/Optimizaci%C3%B3n_de_la_cartera

[55] «Quantum Counting» https://qiskit.org/textbook/ch-algorithms/quantum-counting.html

[56] «Dataset Ad-Hoc»

https://qiskit.org/documentation/machine-learning/stubs/qiskit_machine_learning.datasets.ad_hoc_data.html#qiskit_machine_learning.datasets.ad_hoc_data

[57] «Dataset Heart Failure» https://www.kaggle.com/fedesoriano/heart-failure-prediction

[58] «Scikit-learn» https://es.wikipedia.org/wiki/Scikit-learn

[59] «Sklearn SVC»

https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

[60] «ZZFeatureMap» https://qiskit.org/documentation/stubs/qiskit.circuit.library.ZZFeatureMap.html

[61] «Sklearn PCA»

https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html

[62] «NetWorkX» https://networkx.org/documentation/stable/index.html

[63] «DOcplex» https://ibmdecisionoptimization.github.io/docplex-doc/mp/index.html

[64] «Qiskit NumPyMinimumEigenSolver»

https://qiskit.org/documentation/stubs/qiskit.algorithms.NumPyMinimumEigensolver.html

[65] «Qiskit QAOA» https://qiskit.org/documentation/stubs/qiskit.algorithms.QAOA.html

[66] «Quantum Approximate Optimization Algorithm» https://arxiv.org/pdf/1411.4028.pdf

[67] «Qiskit Finance Portfolio Optimization» https://github.com/Qiskit/qiskit-finance/blob/stable/0.3/docs/tutorials/01_portfolio_optimization.ipynb

[68] «Quandl» https://data.nasdaq.com/

[69] «Qiskit VQE» https://qiskit.org/documentation/stubs/qiskit.algorithms.VQE.html

[70] «NASDAQ» https://es.wikipedia.org/wiki/NASDAQ

[71] P. R. Grasa, «Grover's algorithm on the IBM quantum computers,» http://diposit.ub.edu/dspace/bitstream/2445/180905/1/RODR%c3%8dGUEZ%20GRASA%20PABLO_4253599_assignsubmission_file_TFG-Rodriguez-Grasa-Pablo.pdf

[72] P. V. Zahorodko, «Comparisons of performance between quantum-enhanced and classical machine learning algorithms on the IBM Quantum Experience» https://iopscience.iop.org/article/10.1088/1742-6596/1840/1/012021/meta

[73] A. Wack, «Quality, Speed, and Scale: three key attributes to measure the performance of near-term quantum computers» https://arxiv.org/abs/2110.14108

[74] «CLOPS» https://research.ibm.com/blog/circuit-layer-operations-per-second

[75] «FLOPS» https://en.wikipedia.org/wiki/FLOPS

[76] D. P. John Hennessy, Computer Architecture, November 23, 2017.

[77] «Esfera de Bloch interactuable» https://javafxpert.github.io/grok-bloch/

[78] «Bit vs Qubit» https://telpc.es/img/cms/1_w9516UckuSEBQdiUOoiHbQ.png

[79] «SVM Kernel Trick» https://www.researchgate.net/figure/Kernel-trick-By-transforming-the-original-space-left-into-a-space-of-increased_fig1_305284381

Todas las consultas han sido realizadas entre septiembre de 2021 y enero de 2022