FACULTAD DE ESTUDIOS ESTADÍSTICOS

MÁSTER EN MINERÍA DE DATOS E INTELIGENCIA DE NEGOCIOS

Curso 2020/2021

Trabajo de Fin de Máster

TITULO:

Aplicación práctica de la computación cuántica a problemas financieros

***Alumno*: Carlos García Martínez**

***Tutor*: Ginés Carrascal de las Heras**

Septiembre de 2021

UNIVERSIDAD COMPLUTENSE
MADRID

Resumen

La computación cuántica está experimentando un auge significativo esta última década, lo cual ha permitido a los investigadores plantear problemas que hasta la fecha eran imposibles de resolver en el paradigma clásico.

Los qubits, que son los bits cuánticos, nos dan una ventaja superior respecto a ciertos problemas complejos en distintas ramas, como son la química y la financiera. Más concretamente, en el espectro financiero existen determinados problemas de optimización que su complejidad se va incrementando exponencialmente a medida que añadimos más nodos, por lo que los computadores clásicos encuentran grandes dificultades a la hora de calcular dichos problemas.

Nuestro objetivo es desarrollar un algoritmo cuántico que resuelva el problema del arbitraje de divisas. De tal forma que logremos mejores resultados respecto a los sistemas clásicos que monitorizan actualmente el mercado, analizando así, problemas de tallas mayores con un tiempo menor. Para la realización del trabajo utilizaremos el lenguaje Python y la librería cuántica Qiskit. Además, utilizaremos ciertos computadores cuánticos proporcionados por IBM para ejecutar dichos algoritmos.

Palabras clave: computación cuántica, qubits, arbitraje

Abstract

Quantum computing is experiencing a sector-shifting boom this decade, which has allowed researchers to dive deeper into finding solutions to unsolved problems in the classical paradigm.

Qubits, or quantum bits, bring us many advantages in regard to certain complex problems related to chemicals and finance. Furthermore, in finance there are some optimization problems of deepening complexity with more nodes. This is a result of classical computers finding it difficult to solve problems such as these.

Our objective is to develop a quantum algorithm that solves currencies arbitrage. Moreover, we will achieve better results with this quantum algorithm than with classical systems that are on the market. We will solve big problems with less time using quantum performance. To accomplish this, we will use Python and the quantum library, Qiskit. Moreover, we will use IBM quantum computers to execute these algorithms.

Keywords: quantum computing, qubits, arbitrage

Índice de figuras

Índice de tablas

Índice de contenido

1. Introducción

“Cuántico” es un término que proviene del estudio de la Mecánica Cuántica. Muchas veces, el término "cuántico" se utiliza como palabra de moda en el campo de la física de partículas. Se sabe que la relevancia moderna de que algo sea cuántico proviene del campo de la mecánica cuántica. Este es el campo de la física que aborda las propiedades físicas de la naturaleza a escala atómica.

En nuestro mundo como humanos, experimentamos interacción y fuerzas a una macroescala. A nivel de partículas infinitesimalmente pequeñas (escala subatómica), ya no se aplican los mismos principios de la física que gobiernan nuestra vida diaria. En otras palabras, las reglas se invierten y, a escala cuántica, la mecánica clásica es insuficiente. En un sistema cuántico, todas las cantidades están restringidas a sus valores discretos (como parte de la cuantificación) y los objetos funcionan como partículas y ondas (dualidad onda-partícula). Es fundamental tener en cuenta que, en un entorno cuántico, el valor de una cantidad física se puede predecir antes de su medición si se nos da el conjunto de condiciones iniciales (Principio de incertidumbre de Heisenberg). La mecánica cuántica nació cuando Max Planck propuso su situación al problema de la radiación del cuerpo negro y cuando Einstein presentó su revolucionaria teoría sobre la conexión entre frecuencia y energía (luego explica el efecto fotoeléctrico y la relatividad general) (Sharma, 2020).

1.1 Antecedentes y estado actual del tema

Richard Faynman encendió la chispa de la computación cuántica. En 1981, en el MIT, presentó el siguiente dilema: las computadoras clásicas no pueden simular la evolución de los sistemas cuánticos de manera eficiente. Por lo tanto, propuso un modelo básico para una computadora cuántica que sería capaz de realizar tales simulaciones. Con esto, esbozó la posibilidad de superar exponencialmente a las computadoras clásicas. Sin embargo, pasaron más de 10 años hasta que se creó un algoritmo especial para cambiar la visión de la computación cuántica, el algoritmo de Shor. (Braun, 2018)

En 1994, el actual profesor de Matemáticas del MIT, Peter Shor, que entonces trabajaba en Bell Labs, propuso el ahora conocido como algoritmo de Shor. Este algoritmo cuántico redujo el tiempo de ejecución en factorización numérica de tiempo exponencial a polinomial. Si bien esto puede parecer un problema bastante abstracto, la suposición de que la factorización de enteros es difícil es la base de la seguridad RSA, el principal criptosistema en uso hoy. Como el primer algoritmo en demostrar importante ventaja cuántica para un problema muy importante sin conexión obvia con la mecánica cuántica, el algoritmo de Shor llamó la atención del gobierno, corporaciones y científicos académicos. Esto creó un gran interés en el campo de la información cuántica y dio a los investigadores experimentales una fuerte justificación para desarrollar computadoras cuánticas desde el extremo del hardware.

En 1998, la Universidad de Oxford, seguida poco después por una colaboración entre IBM, UC Berkeley y el MIT Media Lab, ejecutó el algoritmo Deutsch-Jozsa en dispositivos NMR de 2 qubit. Estos sirvieron como las primeras demostraciones de un algoritmo implementado en una computadora cuántica física. (Vasconcelos, 2019)

Más recientemente, en 2019 Google afirmó que había alcanzado la supremacía cuántica. Este término fue definido por John Preskill en 2012 para describir el momento en el cual los sistemas cuánticos pudieran realizar tareas más eficientemente que sus homónimos en el paradigma clásico. (The quantum daily, 2020)

La compañía de Mountain View aseguraba haber conseguido ejecutar en 200 segundos —tres minutos y 20 segundos— una operación para calcular números aleatorios que al superordenador más potente del mundo le hubiera llevado al menos 10.000 años. Esta sería la primera demostración empírica del concepto de supremacía cuántica.

Poco después del anuncio, IBM lo ha puesto en duda, ya que considera que un superordenador basado en computación clásica podría ejecutar el mismo experimento que propone Google en dos días y medio. Ambas compañías son los principales contendientes de la carrera por hacerse con el liderazgo del desarrollo de esta tecnología y han sido los autores de los mayores avances realizados hasta la fecha. (BBVA, 2019)

1.2 Objetivos

En este trabajo nos ocupan diversos objetivos que iremos cumpliendo a medida que se avance con el estudio. El primer objetivo consiste en explicar qué es la computación cuántica y determinar qué beneficios tiene respecto a la computación clásica. De tal forma que explicaremos brevemente los conceptos teóricos y después realizaremos comparaciones entre los distintos paradigmas.

El segundo objetivo es introducir el arbitraje de divisas y explicar mediante ejemplos prácticos cómo se modela este tipo de problemas. Para ello, describiremos ciertos conceptos teóricos referentes a la historia de este tipo de problemas y también explicaremos los grafos que nos ayudan a modelizar los problemas de arbitraje.

Finalmente, nuestro objetivo principal es implementar con Python un algoritmo cuántico que resuelva el problema de arbitraje de divisas y proporcione tiempos de cálculo inferiores a los algoritmos convencionales. Queremos matizar que este algoritmo se hace con perspectivas de futuro, es decir, somos conscientes que el hardware actual no nos ofrecerá una ventaja significativa respecto a la solución en el paradigma clásico. No obstante, esperamos que, en un horizonte temporal de aproximadamente 2 años, la solución cuántica ofrezca ventajas significativas, es decir, tener preparado el software antes del hardware.

1.3 Metodología

La fuente de datos que utilizaremos para extraer la información de las divisas mundiales es ratesapi que es una API gratuita para los tipos de cambio de divisas actuales e históricos publicados por el Banco Central Europeo. Las tarifas se actualizan diariamente a las 15:00 CET. Estos tipos de cambio serán nuestra unidad de análisis.

Nuestro plan de trabajo va a ser el de construir un producto mínimo viable (MVP) e iremos realizando iteraciones sobre este MVP para poder ampliar cada vez más funcionalidades. Todo este trabajo va a realizarse siguiendo metodologías ágiles y buenas prácticas de programación limpia.

2. Computación cuántica

Dentro de la computación cuántica existe un amplio campo que nos llevaría muchas páginas explorarlo con profundidad. No obstante, en este apartado describiremos brevemente los conceptos teóricos desde una visión muy práctica para que el lector entienda los principios que aplicamos en los capítulos posteriores.

2.1 El qubit

El qubit es el análogo cuántico del bit, la unidad fundamental clásica de información. Es un objeto matemático con propiedades específicas que se pueden realizar en un sistema físico real de muchas formas diferentes. Así como el bit clásico tiene un estado (0 o 1), un qubit también tiene un estado. Sin embargo, contrariamente al bit clásico, |0⟩ y |1⟩ son solo dos estados posibles del qubit, y cualquier combinación lineal (superposición) de los mismos también es físicamente posible. Por lo tanto, en general, el estado físico de un qubit es la superposición | ψ⟩ = α |0⟩ + β |1⟩ (donde α y β son números complejos) (The Stanford Encyclopedia of Philosophy, 2019).

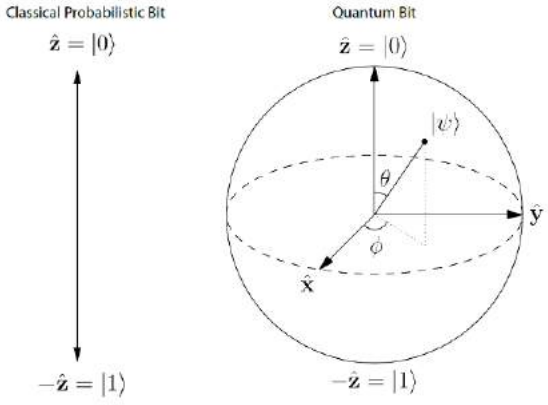

Figura 1: Representación de los estados del bit en ambos paradigmas

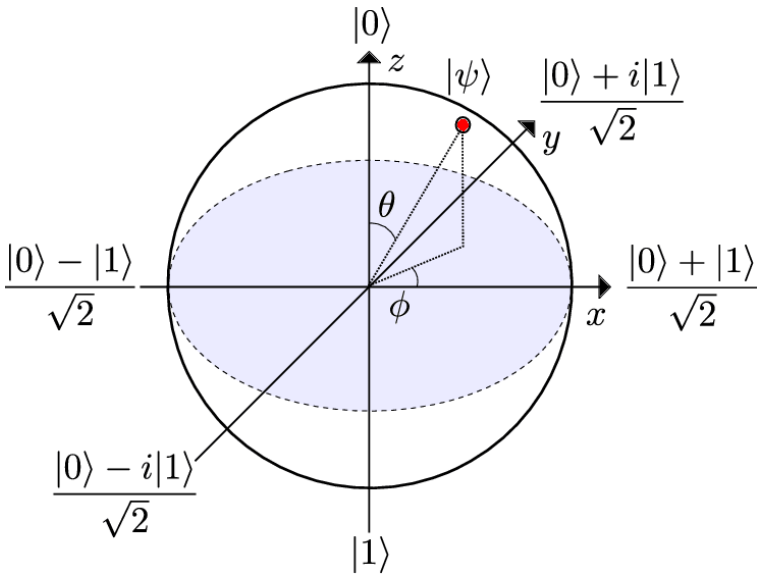

Figura 2: Esfera de Blonch

Para medir un qubit, necesitaremos que este colapse en uno de los dos estados clásicos posibles |0⟩ o |1⟩. Cuando se mide un qubit dado por el estado α |0⟩ + β |1⟩, obtenemos que el 0 resultante con una probabilidad $|\alpha|^2$ y el 1 resultante con una probabilidad $|\beta|^2$ (Microsoft, 2021). Este qubit, dadas sus propiedades matemáticas, podemos representar su estado en una esfera, más conocida como la esfera de Blonch.

En esta esfera, los estados base están colocados en los polos. Las distintas superposiciones de los dos pueden ser convertidas a coordenadas únicas en la esfera.

2.2 Circuitos cuánticos

Ya sea que usemos qubits o bits, debemos manipularlos para convertir las entradas que tenemos en las salidas que necesitamos. Para los programas más simples con muy pocos bits, es útil representar este proceso en un diagrama conocido como diagrama de circuito. Estos tienen entradas a la izquierda, salidas a la derecha y operaciones representadas por símbolos en el medio. Estas operaciones se denominan "puertas".

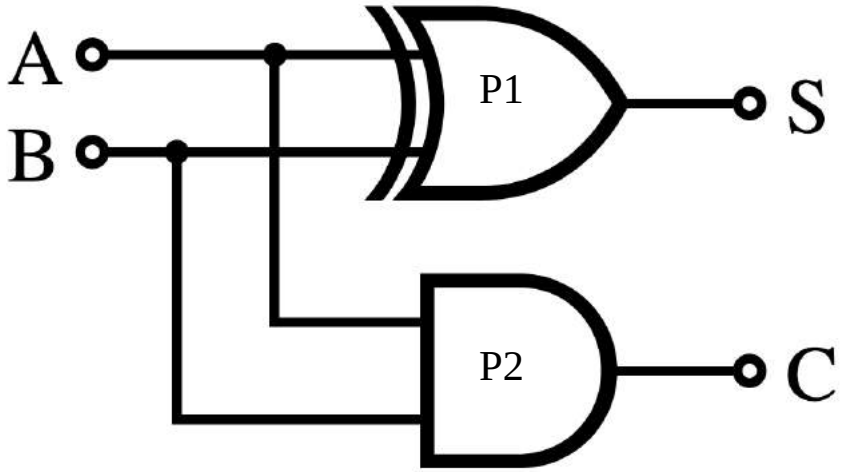

Figura 3: Ejemplo de circuito clásico

En la figura 3 podemos ver que a partir de las entradas A y B, obtenemos las salidas S y C. La puerta 1 es una puerta XOR que implementa el o exclusivo, es decir, una salida verdadera resulta si una, y solo una de las entradas a la puerta es verdadera. Por otra parte, la puerta 2 corresponde a una AND que resulta una salida falsa si alguna de las entradas es falsa. De tal forma que la tabla de la verdad de este circuito será la siguiente:

A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Tabla 1: Tabla de la verdad del circuito clásico

Para las computadoras cuánticas, usamos la misma idea básica, pero tenemos diferentes convenciones sobre cómo representar entradas, salidas y los símbolos usados para las operaciones. Aquí está el circuito cuántico que representa el mismo proceso que el anterior.

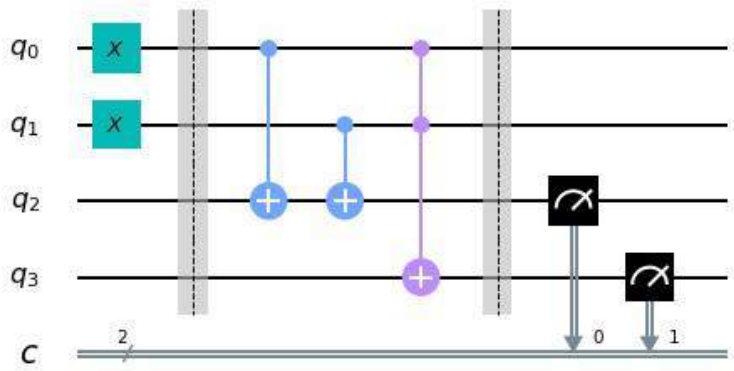

Figura 4: Ejemplo de circuito cuántico

Si nos fijamos en la primera zona delimitada por las barreras, podemos ver que hay 2 puertas NOT, que se utilizan para codificar la entrada. Este caso se utilizaría para replicar el cuarto caso de la tabla de la verdad (donde ambas entradas son 1). Entre las dos barreras, vemos otra zona que servirá para realizar las operaciones que consideremos necesarias, en esta zona podemos ver que hay dos tipos de puertas, dos cnot que corresponde a la notación azul y una puerta Toffoli que corresponde a la notación violeta.

La puerta cnot es una puerta NOT, pero con una importante diferencia, sólo invertirá el bit si el target está a 1. De tal forma que su tabla de verdad será la siguiente:

Input (q1 q0)	Output (q1 q0)
00	00
01	11
10	10
11	01

Tabla 2: Tabla de verdad de la puerta cnot

Por otra parte, la puerta Toffoli que funciona como una puerta AND nos ayudará a calcular el bit de arrastre para cuando la suma sea 1 + 1 = 0. Y este bit de arrastre lo dejará en q3.

Finalmente, en la tercera zona tenemos los medidores que nos ayudarán a saber en qué estado colapsan los qubits. Para ello, utilizaremos un histograma para ver la probabilidad de que sea cierto resultado.

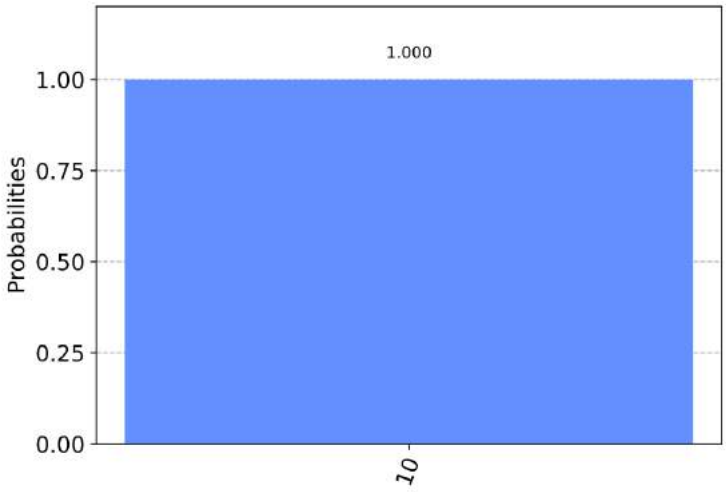

Figura 5: Histograma con los resultados del circuito cuántico half adder

Como podemos ver en el histograma, hay una probabilidad del 100% de que el resultado sea 10. Esto es completamente lógico ya que hemos realizado un circuito muy simple en el cual no hemos modificado la fase de ninguno de los qubits, por lo que no ha habido superposición cuántica. Cuando ejecutemos circuitos más complejos posteriormente en el ordenador cuántico veremos que hay divergencia de resultados (Qiskit Textbook, 2021).

2.3 Puertas cuánticas

En el capítulo anterior hemos podido ver como extrapolar un circuito clásico a un circuito cuántico con dos puertas simples. No obstante, nuestro objetivo al realizar circuitos cuánticos es aprovechar las ventajas de la computación cuántica, de tal forma que debemos de utilizar puertas más complejas que aprovechan estas propiedades. Estas puertas, son matrices que multiplicamos a los vectores que representan el estado de los qubits. Geométricamente, las puertas cuánticas son rotaciones unitarias que aplicamos sobre los distintos ejes de la esfera de Bloch.

Puertas de Pauli (X,Y y Z)

La puerta X, equivale a la puerta clásica NOT. Obteniendo |1> cuando la aplicamos al estado |0>. Tal y como lo vemos en la expresión E1.

$$X|0\rangle = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} = |1\rangle \qquad \text{(E1)}$$

Las puertas Y, Z actúan con las matrices siguientes en nuestro circuito cuántico:

$$Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} \qquad \text{(E2)}$$

$$Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \qquad \text{(E3)}$$

Puerta H

La puerta Handamard gate es una puerta muy utilizada en computación cuántica. Nos permite alejarnos de los polos de la esfera de Bloch y crear una superposición de |0 > y | 1 >. También puede ser expresada como una rotación de 90º en eje y seguida por una rotación de 180 º en el eje X.

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \qquad \text{(E4)}$$

Si nos fijamos en la puerta X, podemos ver que la podemos descomponer en varias puertas más, tal y como se muestra en la figura 6. De tal forma que X = H Z H.

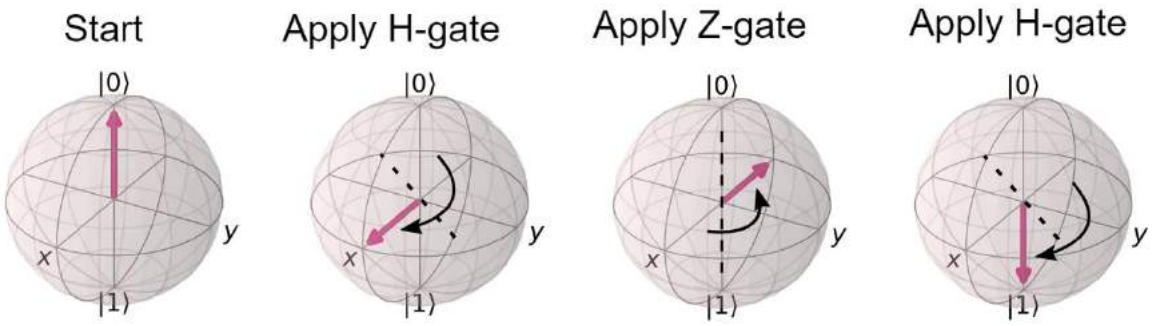

Figura 6: Descomposición de la puerta X

Puerta U

Tal y como hemos demostrado anteriormente, todas las puertas cuánticas se pueden descomponer en otras más generales hasta llegar a la puerta U3 que tiene tres parámetros y puede utilizarse para representar cualquier puerta de un único qubit.

$$U_3(\theta,\phi,\lambda) = \begin{bmatrix} \cos\left(\frac{\theta}{2}\right) & -e^{i\lambda}\sin\left(\frac{\theta}{2}\right) \\ e^{i\phi}\sin\left(\frac{\theta}{2}\right) & e^{i(\phi+\lambda)}\cos\left(\frac{\theta}{2}\right) \end{bmatrix} \qquad \text{(E5)}$$

Donde θ representa el ángulo de rotación frente al eje y, ϕ y λ representan los ángulos de rotación frente al eje z.

2.3 Variational Quantum Eigensolver (VQE)

El algoritmo de estimación de fase (PEA) fue el primer algoritmo propuesto para simular problemas de estructura electrónica en una computadora cuántica. PEA proporciona un camino para obtener la energía electrónica del estado fundamental exacto para una molécula mediante la evolución en el tiempo de un estado cuántico con una superposición significativa con el estado fundamental utilizando el Hamiltoniano molecular de interés (Aspuru-Guzik, 2005).

Debido a las profundidades de circuito muy largas y las puertas cuánticas complejas requeridas por PEA, los tiempos de coherencia necesarios para simular estados electrónicos interesantes excederían los tiempos disponibles en cualquier dispositivo cuántico existente o de corto plazo. Las mejoras al PEA aún requieren recursos significativos y las demostraciones experimentales hasta la fecha tan solo involucran unos qubits (Lanyon, 2010).

Con el fin de reducir las demandas de hardware significativas requeridas por PEA y explotar las capacidades de los dispositivos cuánticos de escala intermedia (Alberto Peruzzo, 2014), VQE fue propuesto y demostrado utilizando qubits fotónicos. Esto fue seguido por varios estudios teóricos sobre VQE y demostraciones en otro hardware como qubits superconductores e iones atrapados (N. Cody Jones, 2012). También se han seguido otros enfoques, incluidos los métodos para la computación cuántica adiabática y el aprendizaje automático cuántico.

En muchas aplicaciones, es importante encontrar el valor propio mínimo de una matriz. Por ejemplo, en química, el valor propio mínimo de una matriz hermitiana que caracteriza a la molécula es la energía del estado fundamental de ese sistema. En el futuro, el algoritmo de estimación de fase cuántica se puede utilizar para encontrar el valor propio mínimo. Sin embargo, su implementación en problemas útiles requiere profundidades de circuito que exceden los límites del hardware disponible en la era NISQ. Así, en 2014, Peruzzo et al. propuso VQE para estimar la energía del estado fundamental de una molécula utilizando circuitos mucho menos profundos (Alberto Peruzzo, 2014).

Enunciado formalmente, dada una matriz hermitiana H con un valor propio mínimo desconocido λ_{min}, asociado con el valor propio $|\psi_{min}\rangle$, VQE proporciona un estimado λ_θ límite λmin:

$$\lambda_{min} \leq \lambda_\theta \equiv \langle \psi(\theta)|H|\psi(\theta)\rangle \qquad \text{(E5)}$$

Donde $|\psi(\theta)\rangle$ es el estado propio asociado con λ_θ. Aplicando un circuito parametrizado, representado por U(θ), a un estado arbitrario $|\psi\rangle$, el algoritmo obtiene un estimado $U(\theta)|\psi\rangle \equiv |\psi(\theta)\rangle$ en $|\psi_{min}\rangle$. La estimación se optimiza iterativamente mediante un controlador clásico que cambia el parámetro θ minimizando el valor estimado de $\langle\psi(\theta)|H|\psi(\theta)\rangle$.

En capítulos posteriores explicaremos cómo aplicar este algoritmo al problema del arbitraje, donde tomaremos una serie de consideraciones que nos ayudarán a modelizar el problema.

3. Arbitraje

3.1 Contexto histórico

Figura 7: Óleo que simboliza el arbitraje en el pasado

¿Qué es el arbitraje? ¿En qué se diferencia el comercio de arbitraje de la especulación? El concepto moderno de "arbitraje" tiene una variedad de definiciones. Muchos libros del mundo de las finanzas la definen como aquella operación de compraventa sin riesgo que genera unas ganancias positivas sin inversión previa. Esta perfecta definición de arbitraje de los mercados se explota en la derivación de fórmulas para reclamos contingentes específicos, como la paridad de interés cubierta y el precio de la opción Black-Scholes, basándose en el supuesto de ausencia de oportunidades de arbitraje. Los estudios sobre los "límites del arbitraje" identifican distorsiones que enturbian la ejecución simultánea de operaciones de arbitraje con connotaciones especulativas: dificultades y costos de "vender en corto" y pedir prestado; retrasos en la ejecución; costos de transacción e información; riesgo de crédito y liquidación; restricciones de capital y así sucesivamente. Estas limitaciones están documentadas en estudios de desempeño empírico que abarcan arbitraje *cash-and-carry*[1], arbitraje de bonos municipales, arbitraje de fusiones, arbitraje estadístico, arbitraje de riesgo y arbitraje de bonos convertibles. Motivado por una diferencia de precio en un momento dado que excede el límite de arbitraje percibido, en la práctica las estrategias a corto y largo plazo involucradas en el comercio de arbitraje requieren algo de capital; se perciben como de bajo riesgo; y tienen una probabilidad limitada de que las ganancias comerciales puedan ser negativas.

Un examen detenido de la ejecución práctica del arbitraje en un momento dado revela detalles sobre la estructura de las instituciones financieras que facilitan las operaciones relevantes. Por ejemplo, considere las operaciones de arbitraje de índices bursátiles japonesas utilizadas para motivar la "sociología del arbitraje" explorada por Miyazaki (2007). El arbitraje está motivado por una diferencia entre el precio de un contrato de futuros sobre índices bursátiles japoneses y el precio del índice bursátil asociado en un momento dado. El arbitrajista reacciona ante la violación momentánea de un límite de arbitraje al intentar acortar simultáneamente lo que es caro y comprar lo que es barato. El uso de contratos de futuros sobre índices bursátiles japoneses y la capacidad de ejecutar rápidamente operaciones en una canasta de acciones que representen o reproduzcan el índice requieren una sofisticación en la estructura de las instituciones financieras que no estuvo disponible hasta finales de los años ochenta. La tecnología de la comunicación y el comercio informatizado facilitan el objetivo de la ejecución simultánea de las posiciones cortas y largas. A su vez, la estructura de las instituciones financieras en períodos históricos anteriores se refleja en restricciones a la capacidad de ejecutar operaciones de arbitraje, acotando los tipos de operaciones que se pueden ejecutar sin cruzar la línea entre el arbitraje y la especulación (Cambridge University Press, 2021).

1. Cash-and-carry: es una estrategia de mercado neutral que combina la compra de una posición larga en un activo como una acción o un producto básico, y la venta (corta) de una posición en un contrato de futuros sobre ese mismo activo subyacente (Investopedia, 2021).

3.2 Arbitraje de divisas en el contexto actual

En nuestro trabajo, nos centraremos en un tipo de arbitraje: el arbitraje de divisas. Este se centra en realizar operaciones de compraventa en distintos brokers o mercados para beneficiarse de las ineficiencias con las discrepancias de precios entre los mismos. Para ilustrarlo de una mejor manera, imagínese dos brokers con diferente cambio US/EUR:

- Plus500: 1,47 USD por EUR
- BDSwiss: 1,39 USD por EUR

El comerciante, por lo tanto, comprará con n euros en Plus 500, obteniendo 1,47 USD por cada euro invertido. Posteriormente, venderá los dólares obtenidos en BD Swiss que le proporcionarán un beneficio de n x 0,08 USD por cada euro invertido. Invirtiendo 500 euros, el comerciante obtendría 40 euros de beneficio.

Hay distintos tipos dentro del arbitraje de divisas en función de cual sea la ventaja que aprovechemos para operar:

- Arbitraje de ubicación: Es el tipo de arbitraje más popular y suele circunscribir dos divisas. El ejemplo previamente explicado corresponde a este tipo.

- Arbitraje triangular: Como sugiere el nombre, este arbitraje implica el uso de tres monedas. Este tipo de arbitraje existe si el tipo de cambio cruzado de dos monedas no coincide con el tipo de cambio de la moneda. En el arbitraje triangular, calculamos el tipo de cambio cruzado de dos monedas y luego lo comparamos con el tipo de cambio real.

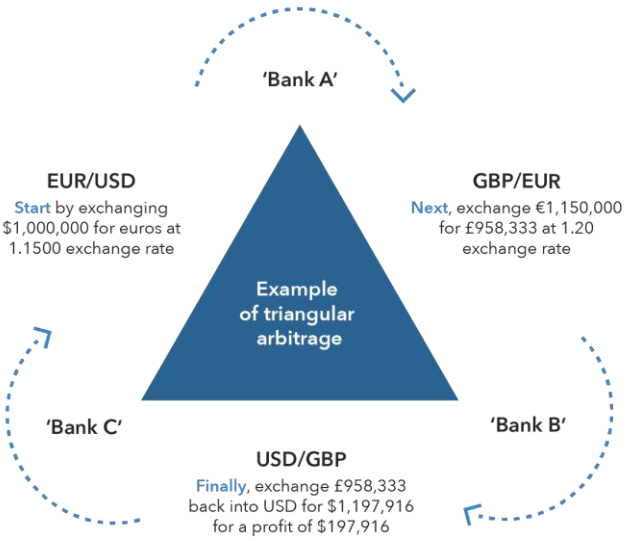

Figura 8: Ejemplo de arbitraje triangular

- Arbitraje de interés cubierto: De acuerdo con esto, un comerciante utiliza diferenciales de tasas de interés para invertir en una moneda. Y, luego, el operador cubre el riesgo con la ayuda de un contrato de futuros o un *contrato forward*[1].

- Arbitraje de spot-futuros: En un arbitraje de este tipo, se produce un intercambio simultáneo de divisas tanto en el mercado al contado como en el de futuros. Por ejemplo, compra USD en el mercado al contado y luego vende el mismo en el mercado de futuros para obtener ganancias si hay alguna irregularidad en los precios (Efinance Management, 2020).

1. Contrato forward: Un forward o contrato a plazo es un acuerdo firme entre dos partes mediante el que se adquiere un compromiso para intercambiar un bien físico o un activo financiero en un futuro, a un precio determinado hoy (Arias, 2017).

3.3 Arbitraje como problema de optimización

Tal y como hemos visto previamente, existen diversos tipos de arbitraje y nuestro problema es una generalización de uno de ellos. Para el arbitraje triangular, participan 3 divisas y calculamos el cambio cruzado de 2 divisas. En nuestro caso, participarán n divisas que conformarán un grafo de precios de cambio tal y como se ilustra en la figura siguiente.

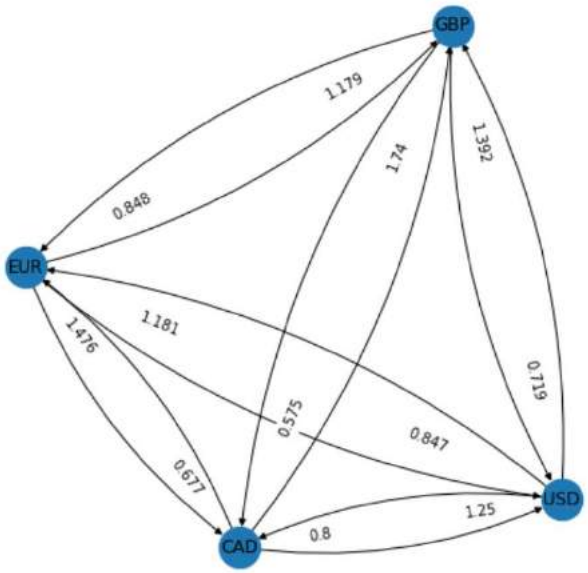

Figura 9: Grafo precios de cambio para las divisas EUR, CAD, GBP y USD

En el grafo dirigido que conforma la figura 9, podemos ver los distintos nodos que conforman las divisas y las aristas son los tipos de cambio que encontramos de una divisa a otra. Nuestro objetivo por lo tanto va a ser encontrar el camino óptimo de un nodo A al mismo nodo A que minimice las comisiones. Hemos de tener en cuenta que cada vez que realizamos un cambio, se generan comisiones de intercambio.

El algoritmo VQE, en un principio se diseñó para la rama de la química, ya que nos ayudaba a calcular en nivel de energía mínimo dada cierta reacción cuántica. I este cálculo, era bastante aproximado a la realidad ya que los qubits pueden simular con bastante precisión los distintos comportamientos a nivel atómico que suceden en dichas reacciones. No obstante, nosotros vamos a utilizar este algoritmo para encontrar el camino que más beneficio nos reporta. Para ello, hemos de modelizar el problema como un algoritmo de optimización híbrido que utiliza tanto el paradigma clásico como el paradigma cuántico. De tal forma que VQE utiliza su parte cuántica para calcular la energía y su parte clásica para optimizar los parámetros.

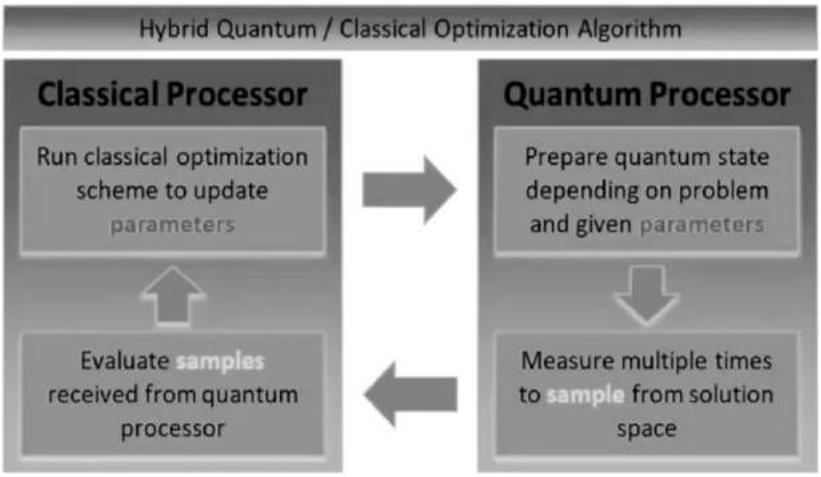

Figura 10: Esquema de hibridación cuántica/clásica de algoritmos de optimización

Para nuestro problema de optimización utilizaremos la optimización binaria cuadrática sin restricciones (QUBO), que corresponde a un problema de optimización combinatoria con una amplia gama de aplicaciones, desde finanzas y economía hasta aprendizaje automático.

QUBO ha sido ampliamente estudiado y se utiliza para modelar y resolver numerosas categorías de problemas de optimización que incluyen instancias importantes de flujos de red, programación, corte máximo, cobertura de vértices y otros problemas de ciencias gráficas y de gestión, integrándolos en un marco de modelado unificado (Lewis, 2017).

Nuestro problema, por tanto, debe cumplir 3 características:

- Variables de optimización binaria: $x_i \in \{0,1\}$
- Cuadrático: $x_i * x_j$
- Sin restricciones: $\min_{x \in \{0,1\}^n} x^T Q x + c^T x$

No obstante, para la última restricción debemos tener en cuenta que nuestro problema sí tiene restricciones ya que por ejemplo la solución debe ser un círculo cerrado. Es decir, debemos empezar por sustituir cada restricción por un término cuadrático. Esto hará que la penalización sea muy grande y por lo tanto hará que el algoritmo no rebase estos límites:

$$\left(\mathrm{a}^{\mathrm{T}} x - b\right)^2$$

De tal forma que el problema se quedará planteado de la siguiente forma:

$$\max_{x \in \{0,1\}^{n \times n}} x\Sigma$$

Donde usamos la siguiente notación:

- $x \in \{0,1\}^{nxn}$ denota la matriz de variables de decisión binarias, que indican si elegimos el arista (x[i] = 1) o si no lo elegimos (x[i] = 0)
- $\Sigma \in \mathbb{R}_{n \times n}$ especifica la tasa de cambio de las diferentes divisas (transit_price_matrix)
- L denota la longitud máxima de la ruta, es decir, el número de intercambios que se realizarán para cerrar el ciclo

Además, asumimos la siguiente simplificación:

- todos los activos tienen el mismo valor intrínseco (normalizado a 1)

Hemos de recalcar que este problema estará sujeto a una serie de restricciones que explicaremos con detalle en el siguiente apartado.

4. Modelización del problema de arbitraje con hibridación cuántica/clásica

En este apartado explicaremos cómo se ha implementado paso a paso el problema y los resultados que nos ha calculado el script. Incluiremos ciertos fragmentos del código Python utilizado en las explicaciones que van a continuación. Por otra parte, si el lector desea profundizar más, encontrará en el anexo el código completo.

4.1 Consideraciones previas

En primer lugar, debemos definir las restricciones de optimización que definirán el modelo.

1. Condición de arbitraje de círculo cerrado: Para cada divisa, el tránsito de entrada es igual al tránsito de salida
2. Cada divisa solo puede ser transitada una vez
3. Toda la ruta de arbitraje debe ser menor que una longitud determinada

```
In [44]: # 1. closed-circle arbitrage requirement, for each currency, transit-in equals transit-out.
mdl.add_constraints(
    mdl.sum(x[currency, :]) == mdl.sum(x[:, currency]) for currency in range(length))

# 2. each currency can only be transited-in at most once.
mdl.add_constraints(mdl.sum(x[currency, :]) <= 1 for currency in range(length))

# 3. the whole arbitrage path should be less than a given length
path_length = None
if isinstance(path_length, int):
    mdl.add_constraint(mdl.sum(x) <= path_length)
```

Figura 11: Implementación de las restricciones de optimización

Por otra parte, hemos de tener en cuenta las comisiones que se generan cuando intercambiamos divisas. Es decir, cuantos más cambios hagamos será peor ya que tendremos que pagar más comisiones y eso reducirá significativamente nuestras oportunidades de beneficio. Para modelizar estas comisiones, hemos supuesto que por cada operación nos cobrarán un 2% de nuestra operación, ya que, lo normal se encuentra en un rango del 1 al 3 por ciento en cada operación que realizamos (Investopedia, 2021).

Esto lo podemos aplicar a la matriz donde están las tasas de cambio. No lo hemos implementado en nuestro proyecto ya que esta aproximación a la realidad necesitaría de grafos más complejos que proporcionaran mayores tasas de beneficio. En nuestro estudio utilizamos tallas pequeñas por lo que si además de obtener poco beneficio sumamos una cifra de comisiones tan elevada, no encontraríamos caminos óptimos que nos generaran beneficio.

Además, hemos de tener en cuenta que vamos a hacer operaciones con matrices muy grandes y que estas por lo tanto van a utilizar bastante memoria. Así que debemos pensar una forma de aprovechar los recursos de nuestra máquina. Si relajamos la suposición de que todos los coeficientes son números enteros, podemos aproximar los no enteros dividiendo todos los valores en μ y Σ por el

mayor y aproximando cada valor k por una fracción k con 2m − 1 k <2m − 1, donde m es el número de valor qubits (arXiv:1912.04088 [quant-ph] Cornell University, 2021).

Hemos decidido que vamos a operar con matrices de números enteros en vez de matrices con números de coma flotante. En primer lugar, utilizando la librería forex_python vamos a descargar los tipos de cambio actuales en forma de dataframe[1].

	GBP	EUR	USD	CHF	CAD
GBP	0.000000	0.8506	0.717806	0.792509	0.574070
EUR	1.175641	0.0000	0.843882	0.931706	0.674900
USD	1.393134	1.1850	0.000000	1.104072	0.799757
CHF	1.261815	1.0733	0.905738	0.000000	0.724371
CAD	1.741947	1.4817	1.250380	1.380509	0.000000

Figura 12: Tipos de cambio para los diferentes tipos de divisas

Posteriormente, escalaremos al máximo, dividiendo cada elemento del dataframe entre 1.741947 que es el valor máximo correspondiente a esta matriz. Una vez realizada esta transformación, multiplicamos por 2^5 ya que hemos supuesto que nuestro procesador cuántico es de 5 qubits.

	GBP	EUR	USD	CAD
GBP	0.0	15.0	13.0	10.0
EUR	21.0	0.0	15.0	12.0
USD	25.0	21.0	0.0	14.0
CAD	32.0	27.0	22.0	0.0

Figura 13: Tipos de cambio con la transformación semicompleta

Finalmente, podemos observar que en la matriz aún no tenemos enteros, por lo tanto tendremos que hacer un *casting* explícito que convierte todas las columnas del dataframe a int. En el código mostrado a continuación podemos ver cómo se ha realizado toda la transformación.

```
In [15]: df.dtypes
         ##Transformation of the df to integers
         old_df = df.copy()
         ##Scale to the maximum
         maximum = 0
         qubits = 5
         for index, row in df.iterrows(): #Cuidado, complejidad n^2
             for x in currency_set:
                 maximum = max(maximum,float(row[x]))

         print(maximum)
         ###Divide by the maximum
         for index, row in df.iterrows():
             for x in currency_set:
                 row[x] = (float(row[x]) / float(maximum))
                 row[x] = int(row[x] * 2**qubits)
                 row[x] = row[x].astype(int)
         #Change df types
         df = df.astype('int')
```

Figura 14: Fragmento de código en el cual se transforma la matriz de números en coma flotante a enteros

Con las transformaciones realizadas, obtenemos el grafo que se muestra a continuación:

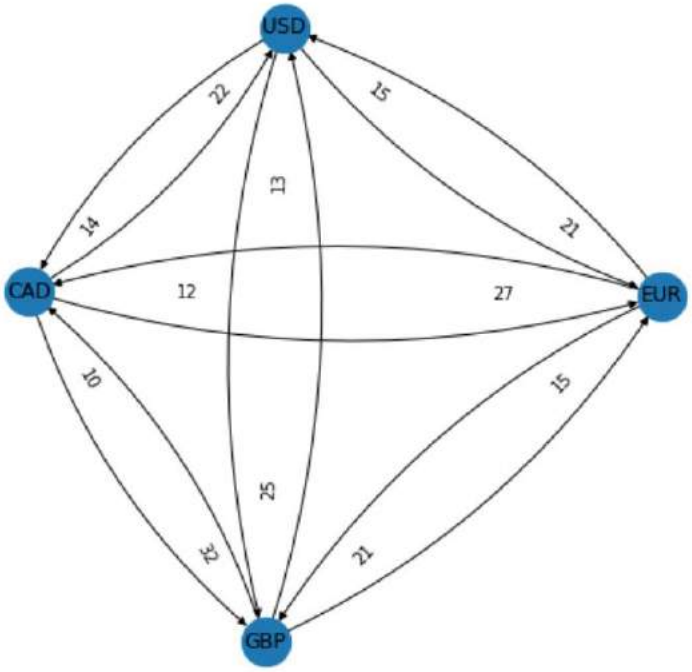

Figura 15: Grafo precios de cambio con la transformación a enteros

Si comparamos con la figura 9 podemos ver que a pesar de que las aristas sean los mismos en este grafo dirigido, los pesos ahora tienen valores enteros, lo que facilitará en un futuro las operaciones a realizar.

4.2 Implementación

En este apartado vamos a explicar cómo hemos programado este algoritmo mediante la librería Qiskit escrita en Python. Vamos a subdividir este apartado en dos perspectivas, en primer lugar, vamos a implementar el algoritmo desde el enfoque clásico y posteriormente vamos a implementar dicho algoritmo desde el enfoque cuántico. Este contraste va a realizarse para comprobar las diferencias entre ambos paradigmas y determinar por qué el cuántico es mejor para este tipo de problemas.

4.2.1 Enfoque clásico

Debemos importar la librería docplex que nos facilitará el modelado de problemas de optimización. Esta librería esta compuesta por dos módulos:

- Mathematical Programming Modeling for Python using docplex.mp (DOcplex.MP)
- Constraint Programming Modeling for Python using docplex.cp (DOcplex.CP)

En la figura que observamos a continuación podemos ver cómo docplex ha modelizado el problema de optimización siguiendo unas directrices matemáticas. En primer lugar donde está el apartado que dice Maximize, está la función objetivo que vamos a maximizar, esto irá determinado por los parámetros x_0,x_1,x_2…x_11 que calcularemos en una etapa posterior. Estos parámetros están sujetos a unas restricciones que se muestran en “Subject to” y se nos proporcionan las inecuaciones y ecuaciones que deben de cumplir los parámatreos para que sean considerados válidos. Por último, en el apartado de bounds, encontramos los límites a los cuales están sometidos dichos parámetros.

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

Maximize
 obj: 2.708050201102 x_0 + 2.564949357462 x_1 + 2.302585092994 x_2
      + 3.044522437723 x_3 + 2.708050201102 x_4 + 2.484906649788 x_5
      + 3.218875824868 x_6 + 3.044522437723 x_7 + 2.639057329615 x_8
      + 3.465735902800 x_9 + 3.295836866004 x_10 + 3.091042453358 x_11
Subject To
 c0: x_0 + x_1 + x_2 - x_3 - x_6 - x_9 = 0
 c1: - x_0 + x_3 + x_4 + x_5 - x_7 - x_10 = 0
 c2: - x_1 - x_4 + x_6 + x_7 + x_8 - x_11 = 0
 c3: - x_2 - x_5 - x_8 + x_9 + x_10 + x_11 = 0
 c4: x_0 + x_1 + x_2 <= 1
 c5: x_3 + x_4 + x_5 <= 1
 c6: x_6 + x_7 + x_8 <= 1
 c7: x_9 + x_10 + x_11 <= 1

Bounds
 0 <= x_0 <= 1
 0 <= x_1 <= 1
 0 <= x_2 <= 1
 0 <= x_3 <= 1
 0 <= x_4 <= 1
 0 <= x_5 <= 1
 0 <= x_6 <= 1
 0 <= x_7 <= 1
 0 <= x_8 <= 1
 0 <= x_9 <= 1
 0 <= x_10 <= 1
 0 <= x_11 <= 1

Binaries
 x_0 x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9 x_10 x_11
End
```

Figura 16: Modelo docplex

Finalmente, deberemos de habilitar aqua_globals.massive para permitir el uso estructuras de datos muy grandes. Una vez lo ejecutemos podremos comprobar que se hace un uso exhaustivo de la memoria lo cual discutiremos en el apartado de resultados.

```
'''
ValueError: 'to_matrix' will return an exponentially large vector,
in this case '16777216' elements. Set aqua_globals.massive=True
or the method argument massive=True if you want to proceed.
'''
aqua_globals.massive=True

# solve classically as reference
opt_result = MinimumEigenOptimizer(NumPyMinimumEigensolver()).solve(qp)
opt_result
```

Figura 17: Fragmento de código para resolver clásicamente

4.2.1 Enfoque cuántico

Para convertir nuestro problema de optimización a un enfoque cuántico, debemos eliminar las inecuaciones que hemos visto antes ya que no son compatibles con el optimizador cuántico y pueden generarnos problemas a la hora de ejecutar el modelo. Para resolver esto, hemos transformado las inecuaciones en ecuaciones, que equivalen a las mismas restricciones previamente planteadas.

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

Maximize
 obj: 2.708050201102 x_0 + 2.564949357462 x_1 + 2.302585092994 x_2
      + 3.044522437723 x_3 + 2.708050201102 x_4 + 2.484906649788 x_5
      + 3.218875824868 x_6 + 3.044522437723 x_7 + 2.639057329615 x_8
      + 3.465735902800 x_9 + 3.295836866004 x_10 + 3.091042453358 x_11
Subject To
 c0: x_0 + x_1 + x_2 - x_3 - x_6 - x_9 = 0
 c1: - x_0 + x_3 + x_4 + x_5 - x_7 - x_10 = 0
 c2: - x_1 - x_4 + x_6 + x_7 + x_8 - x_11 = 0
 c3: - x_2 - x_5 - x_8 + x_9 + x_10 + x_11 = 0
 c4: x_0 + x_1 + x_2 + c4@int_slack = 1
 c5: x_3 + x_4 + x_5 + c5@int_slack = 1
 c6: x_6 + x_7 + x_8 + c6@int_slack = 1
 c7: x_9 + x_10 + x_11 + c7@int_slack = 1

Bounds
 0 <= x_0 <= 1
 0 <= x_1 <= 1
 0 <= x_2 <= 1
 0 <= x_3 <= 1
 0 <= x_4 <= 1
 0 <= x_5 <= 1
 0 <= x_6 <= 1
 0 <= x_7 <= 1
 0 <= x_8 <= 1
 0 <= x_9 <= 1
 0 <= x_10 <= 1
 0 <= x_11 <= 1
       c4@int_slack <= 1
       c5@int_slack <= 1
       c6@int_slack <= 1
       c7@int_slack <= 1

Binaries
 x_0 x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9 x_10 x_11

Generals
 c4@int_slack c5@int_slack c6@int_slack c7@int_slack
End
```

Figura 18: Modelo docplex sin las inecuaciones

Posteriormente, transformamos todas las variables a binarias mediante la función IntegerToBinary(). El resultado lo podemos ver en la figura a continuación.

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

Maximize
 obj: 2.708050201102 x_0 + 2.564949357462 x_1 + 2.302585092994 x_2
      + 3.044522437723 x_3 + 2.708050201102 x_4 + 2.484906649788 x_5
      + 3.218875824868 x_6 + 3.044522437723 x_7 + 2.639057329615 x_8
      + 3.465735902800 x_9 + 3.295836866004 x_10 + 3.091042453358 x_11
Subject To
 c0: x_0 + x_1 + x_2 - x_3 - x_6 - x_9 = 0
 c1: - x_0 + x_3 + x_4 + x_5 - x_7 - x_10 = 0
 c2: - x_1 - x_4 + x_6 + x_7 + x_8 - x_11 = 0
 c3: - x_2 - x_5 - x_8 + x_9 + x_10 + x_11 = 0
 c4: x_0 + x_1 + x_2 + c4@int_slack@0 = 1
 c5: x_3 + x_4 + x_5 + c5@int_slack@0 = 1
 c6: x_6 + x_7 + x_8 + c6@int_slack@0 = 1
 c7: x_9 + x_10 + x_11 + c7@int_slack@0 = 1

Bounds
 0 <= x_0 <= 1
 0 <= x_1 <= 1
 0 <= x_2 <= 1
 0 <= x_3 <= 1
 0 <= x_4 <= 1
 0 <= x_5 <= 1
 0 <= x_6 <= 1
 0 <= x_7 <= 1
 0 <= x_8 <= 1
 0 <= x_9 <= 1
 0 <= x_10 <= 1
 0 <= x_11 <= 1
 0 <= c4@int_slack@0 <= 1
 0 <= c5@int_slack@0 <= 1
 0 <= c6@int_slack@0 <= 1
 0 <= c7@int_slack@0 <= 1

Binaries
 x_0 x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9 x_10 x_11 c4@int_slack@0 c5@int_slack@0
 c6@int_slack@0 c7@int_slack@0
End
```

Figura 19: Modelo docplex con las variables a binario

Finalmente, hemos de crear un modelo sin restricciones. Para ello pondremos todas las restricciones en la función objetivo. Además, debemos incorporar una penalización que será la talla del problema.

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

Maximize
 obj: 10.708050201102 x_0 + 10.564949357462 x_1 + 10.302585092994 x_2
      + 11.044522437723 x_3 + 10.708050201102 x_4 + 10.484906649788 x_5
      + 11.218875824868 x_6 + 11.044522437723 x_7 + 10.639057329615 x_8
      + 11.465735902800 x_9 + 11.295836866004 x_10 + 11.091042453358 x_11
      + 8 c4@int_slack@0 + 8 c5@int_slack@0 + 8 c6@int_slack@0
      + 8 c7@int_slack@0 + [ - 24 x_0^2 - 32 x_0*x_1 - 32 x_0*x_2 + 32 x_0*x_3
      + 16 x_0*x_4 + 16 x_0*x_5 + 16 x_0*x_6 - 16 x_0*x_7 + 16 x_0*x_9
      - 16 x_0*x_10 - 16 x_0*c4@int_slack@0 - 24 x_1^2 - 32 x_1*x_2 + 16 x_1*x_3
      - 16 x_1*x_4 + 32 x_1*x_6 + 16 x_1*x_7 + 16 x_1*x_8 + 16 x_1*x_9
      - 16 x_1*x_11 - 16 x_1*c4@int_slack@0 - 24 x_2^2 + 16 x_2*x_3 - 16 x_2*x_5
      + 16 x_2*x_6 - 16 x_2*x_8 + 32 x_2*x_9 + 16 x_2*x_10 + 16 x_2*x_11
      - 16 x_2*c4@int_slack@0 - 24 x_3^2 - 32 x_3*x_4 - 32 x_3*x_5 - 16 x_3*x_6
      + 16 x_3*x_7 - 16 x_3*x_9 + 16 x_3*x_10 - 16 x_3*c5@int_slack@0 - 24 x_4^2
      - 32 x_4*x_5 + 16 x_4*x_6 + 32 x_4*x_7 + 16 x_4*x_8 + 16 x_4*x_10
      - 16 x_4*x_11 - 16 x_4*c5@int_slack@0 - 24 x_5^2 + 16 x_5*x_7 - 16 x_5*x_8
      + 16 x_5*x_9 + 32 x_5*x_10 + 16 x_5*x_11 - 16 x_5*c5@int_slack@0
      - 24 x_6^2 - 32 x_6*x_7 - 32 x_6*x_8 - 16 x_6*x_9 + 16 x_6*x_11
      - 16 x_6*c6@int_slack@0 - 24 x_7^2 - 32 x_7*x_8 - 16 x_7*x_10
      + 16 x_7*x_11 - 16 x_7*c6@int_slack@0 - 24 x_8^2 + 16 x_8*x_9
      + 16 x_8*x_10 + 32 x_8*x_11 - 16 x_8*c6@int_slack@0 - 24 x_9^2
      - 32 x_9*x_10 - 32 x_9*x_11 - 16 x_9*c7@int_slack@0 - 24 x_10^2
      - 32 x_10*x_11 - 16 x_10*c7@int_slack@0 - 24 x_11^2
      - 16 x_11*c7@int_slack@0 - 8 c4@int_slack@0^2 - 8 c5@int_slack@0^2
      - 8 c6@int_slack@0^2 - 8 c7@int_slack@0^2 ]/2 -16
Subject To

Bounds
 0 <= x_0 <= 1
 0 <= x_1 <= 1
 0 <= x_2 <= 1
 0 <= x_3 <= 1
 0 <= x_4 <= 1
 0 <= x_5 <= 1
 0 <= x_6 <= 1
 0 <= x_7 <= 1
 0 <= x_8 <= 1
 0 <= x_9 <= 1
 0 <= x_10 <= 1
 0 <= x_11 <= 1
 0 <= c4@int_slack@0 <= 1
 0 <= c5@int_slack@0 <= 1
 0 <= c6@int_slack@0 <= 1
 0 <= c7@int_slack@0 <= 1

Binaries
 x_0 x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9 x_10 x_11 c4@int_slack@0 c5@int_slack@0
 c6@int_slack@0 c7@int_slack@0
End
```

Figura 20: Modelo docplex final

Finalmente, ya tenemos nuestro modelo preparado para ejecutar en un entorno cuántico. Utilizaremos este fragmento de código para configurar la ejecución:

```
In [119]: # set classical optimizer
          maxiter = 100
          optimizer = COBYLA(maxiter=maxiter)

          # set variational ansatz
          var_form = RealAmplitudes(length, reps=1)
          m = var_form.num_parameters

          # set backend
          backend_name = 'qasm_simulator'  # use this for QASM simulator
          # backend_name = 'statevector_simulator'  # use this for statevector simlator
          backend = Aer.get_backend(backend_name)
```

Figura 21: Configuración de la ejecución cuántica

Si nos fijamos en la figura anterior, el primer parámetro "maxiter" hace referencia al número máximo de iteraciones que va a realizar el algoritmo para que se realice la convergencia de parámetros. Después tenemos el optimizador.

ADAM	Optimizadores Adam y AMSGRAD.
AQGD	Descenso de gradiente cuántico analítico (AQGD) con optimizador Epochs.
CG	Optimizador de gradiente conjugado.
COBYLA	Optimización restringida por optimizador de aproximación lineal.
L_BFGS_B	Optimizador BFGS Bound de memoria limitada.
GSLS	Búsqueda de líneas suavizadas por Gauss.
NELDER_MEAD	Optimizador de Nelder-Mead.
NFT	Algoritmo Nakanishi-Fujii-Todo.
P_BFGS	Optimizador BFGS paralelo de memoria limitada.
POWELL	Optimizador Powell
SLSQP	Optimizador de programación secuencial de mínimos cuadrados.
SPSA	Optimizador de Aproximación Estocástica de Perturbación Simultánea (SPSA).
TNC	Optimizador de Newton truncado (TNC)

Tabla 3: Tipos de optimizadores locales en qiskit

Qiskit aqua contiene una gran variedad de optimizadores clásicos que se pueden utilizar con algoritmos variacionales cuánticos como por ejemplo VQE. Lógicamente estos optimizadores pueden dividirse en dos categorías:

- Optimizadores locales: Dado un problema de optimización, un optimizador local es una función que intenta encontrar un valor óptimo dentro del conjunto vecino de una solución candidata.
- Optimizadores globales: Dado un problema de optimización, un optimizador global es una función que intenta encontrar un valor óptimo entre todas las soluciones posibles.

CRS	Búsqueda aleatoria controlada (CRS) con optimizador de mutación local.
DIRECT_L	Proporcionar RECTangles Optimizador con polarización local.
DIRECT_L_RAND	Optimizador aleatorio con polarización local Dividing RECTangles
ESCH	Optimizador evolutivo ESCH.
ISRES	Optimizador de estrategia de evolución de ranking estocástico mejorado.

Tabla 4: Tipos de optimizadores globales en qiskit

Posteriormente, nos da la opción de configurar el *ansatz.* Un *ansatz* es la solución aproximada al problema que vamos a resolver. En última instancia lo que va a hacer el optimizador es expresar este ansatz en función de ciertos parámetros que va a ir cambiando hasta conseguir la solución que más se aproxime a nuestro problema.

VQE, por lo tanto, calculará en modo cuántico la función de coste, que será multiplicar el hamiltoniano por el ansatz. Al final, realizamos la multiplicación de una matriz por un vector, que en el paradigma clásico crece exponencialmente. De tal forma que, utilizamos el optimizador clásico que pone los parámetros del ansatz, y luego calculamos la función en el procesador cuántico y le damos el valor resultante al optimizador, para que calcule los siguientes parámetros del ansatz, así hasta que llega a los parámetros que convierten al ansatz en la solución o al menos una que se le parece mucho.

Por tanto, es importante la elección del optimizador clásico que vamos a utilizar ya que, cuanto más ajustemos el ansatz a la solución, menos tiempo utilizará VQE para encontrar la solución óptima.

Finalmente, encontramos el 'backend' esto se refiere al entorno donde vamos a ejecutar nuestro programa cuántico. El que tenemos seleccionado en nuestro ejemplo es el QASM simulator, que simula un ordenador cuántico perfecto en nuestro pc personal. Sin embargo, para este estudio IBM nos ha proporcionado la herramienta Quantum Lab que es una interfaz web donde podemos enviar nuestros programas cuánticos para su ejecución.
Como podemos ver en la figura, hay bastantes computadores cuánticos a nuestra disposición y tenemos una descripción muy detallada de cada uno de ellos. Para determinar cuales son mejores, nos fijaremos en 3 características:

- Número de qubits: Determina la capacidad de cálculo que tiene el ordenador cuántico. Cuantos más qubits más poder computacional.
- Volumen cuántico: El volumen cuántico es una métrica vinculada al rendimiento de las capacidades e índices de error de un ordenador cuántico. A mayor volumen cuántico, menor índice de error y por lo tanto mejor es nuestro ordenador cuántico
- Distancia al computador: Esta también es una característica a tener en cuenta, aunque es menos importante que las otras dos. La distancia al computador influye en términos de latencia ya que cuanto más lejos esté, más retraso tendremos en nuestras órdenes para enviar y/o recibir trabajos.

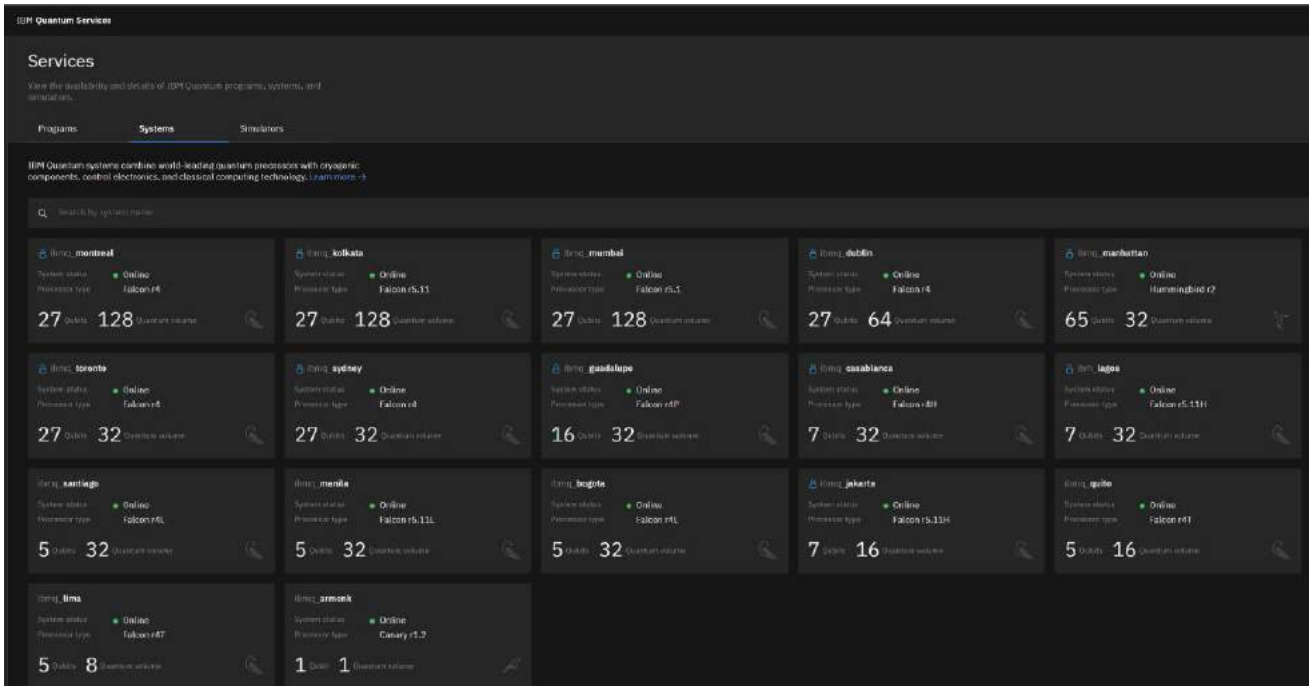

Figura 22: Lista de computadores disponibles en Quantum Lab

Podemos ver en la figura anterior que hay algunos computadores con un candado azul, esto quiere decir que no podemos utilizarlos de forma gratuita. Tendríamos que obtener una licencia de pago para poder utilizar estos computadores con prestaciones superiores.

Volviendo al código, cuando inicializamos VQE se nos muestra la explicación de los parámetros que hemos configurado previamente.

```
==================== Setting of VQE ============================
Algorithm: VQE
-- max_evals_grouped: 1
-- circuit_sampler: <qiskit.aqua.operators.converters.circuit_sampler.CircuitSampler object at 0x000002008016CCD0>
-- expectation: None
-- user_valid_expectation: False
-- include_custom: False
-- expect_op: None
-- operator: None
-- quantum_instance:
Qiskit Terra version: 0.16.1
Backend: 'qasm_simulator (AerProvider)', with following setting:
{'basis_gates': ['u1', 'u2', 'u3', 'u', 'p', 'r', 'rx', 'ry', 'rz', 'id', 'x', 'y', 'z', 'h', 's', 'sdg', 'sx', 't', 'tdg', 'sw
ap', 'cx', 'cy', 'cz', 'csx', 'cp', 'cu1', 'cu2', 'cu3', 'rxx', 'ryy', 'rzz', 'rzx', 'ccx', 'cswap', 'mcx', 'mcy', 'mcz', 'mcs
x', 'mcp', 'mcu1', 'mcu2', 'mcu3', 'mcrx', 'mcry', 'mcrz', 'mcr', 'mcswap', 'unitary', 'diagonal', 'multiplexer', 'initialize',
'kraus', 'roerror', 'delay'], 'coupling_map': None}
{'initial_layout': None, 'seed_transpiler': None, 'optimization_level': None}
RunConfig(max_credits=10, shots=1024)
{'timeout': None}
{}
{}
Measurement mitigation: None
-- optimizer: <qiskit.aqua.components.optimizers.cobyla.COBYLA object at 0x0000020080039EE0>
-- gradient: None
-- cost_fn: <bound method VQE._energy_evaluation of <qiskit.aqua.algorithms.minimum_eigen_solvers.vqe.VQE object at 0x000002008
093AF40>>
-- initial_point: Random seed
-- var_form:
q_0: ┤ RY(θ[0]) ├──■────■────■──┤ RY(θ[4]) ├──────────────────────
q_1: ┤ RY(θ[1]) ├──X────┼────┼──────────────┤ RY(θ[5]) ├──────────
q_2: ┤ RY(θ[2]) ├───────X────X──────────────────────────┤ RY(θ[6]) ├
q_3: ┤ RY(θ[3]) ├──────────────X────X──────────X────────┤ RY(θ[7]) ├
-- var_form_params: [Parameter(θ[0]), Parameter(θ[1]), Parameter(θ[2]), Parameter(θ[3]), Parameter(θ[4]), Parameter(θ[5]), Para
meter(θ[6]), Parameter(θ[7])]
-- parameterized_circuits: None
-- ret: None
-- eval_time: None
-- callback: None
-- aux_op_nones: []
-- aux_operators: []
-- eval_count: 0
================================================================
```

Figura 23: Configuración de los parámetros de VQE

Se puede apreciar en la figura anterior que qiskit nos ha dibujado el circuito cuántico que hemos creado con VQE. Tenemos que para los 4 qubits se han creado distintas puertas Ry donde θ indica el grado de rotación sobre el eje y. Y su representación matricial para θ = 0 es la siguiente:

$$RY(\theta) = exp(-i\frac{\theta}{2}Y) = \begin{pmatrix} \cos\frac{\theta}{2} & -\sin\frac{\theta}{2} \\ \sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{pmatrix}$$

Por otra parte, tenemos la CNOT, que se denota en el esquema por una x y un cuadrado. Esta puerta da la vuelta al qubit objetivo (el que se encuentra en la x) si el qubit de control es 1. En cambio, si el qubit de control es 0 no se realiza ningún cambio sobre el qubit objetivo (Véase Tabla 2).

Una vez ya tenemos nuestro algoritmo inicializado, procedemos a ejecutarlo, y empezaremos a ver como va realizando iteraciones para calcular los parámetros óptimos.

```
2021-08-13 18:39:06,934:qiskit.aqua.operators.converters.circuit_sampler:DEBUG: Parameter conversion 0.00000 (ms)
2021-08-13 18:39:09,302:qiskit.aqua.algorithms.minimum_eigen_solvers.vqe:INFO: Energy evaluation returned [-32.58481194] - 236
8.71362 (ms), eval count: 92
2021-08-13 18:39:09,302:qiskit.aqua.operators.converters.circuit_sampler:DEBUG: Parameter conversion 0.00000 (ms)
2021-08-13 18:39:11,255:qiskit.aqua.algorithms.minimum_eigen_solvers.vqe:INFO: Energy evaluation returned [-33.08369353] - 195
2.55351 (ms), eval count: 93
2021-08-13 18:39:11,255:qiskit.aqua.operators.converters.circuit_sampler:DEBUG: Parameter conversion 0.00000 (ms)
2021-08-13 18:39:13,291:qiskit.aqua.algorithms.minimum_eigen_solvers.vqe:INFO: Energy evaluation returned [-32.54166481] - 203
6.51786 (ms), eval count: 94
2021-08-13 18:39:13,311:qiskit.aqua.operators.converters.circuit_sampler:DEBUG: Parameter conversion 19.21463 (ms)
2021-08-13 18:39:15,303:qiskit.aqua.algorithms.minimum_eigen_solvers.vqe:INFO: Energy evaluation returned [-31.94496822] - 201
1.17611 (ms), eval count: 95
2021-08-13 18:39:15,303:qiskit.aqua.operators.converters.circuit_sampler:DEBUG: Parameter conversion 0.00000 (ms)
2021-08-13 18:39:17,230:qiskit.aqua.algorithms.minimum_eigen_solvers.vqe:INFO: Energy evaluation returned [-32.46470835] - 192
7.64044 (ms), eval count: 96
2021-08-13 18:39:17,230:qiskit.aqua.operators.converters.circuit_sampler:DEBUG: Parameter conversion 0.00000 (ms)
2021-08-13 18:39:19,069:qiskit.aqua.algorithms.minimum_eigen_solvers.vqe:INFO: Energy evaluation returned [-33.45480151] - 183
9.11514 (ms), eval count: 97
2021-08-13 18:39:19,069:qiskit.aqua.operators.converters.circuit_sampler:DEBUG: Parameter conversion 0.00000 (ms)
2021-08-13 18:39:21,432:qiskit.aqua.algorithms.minimum_eigen_solvers.vqe:INFO: Energy evaluation returned [-30.5157612] - 2362.
48732 (ms), eval count: 98
2021-08-13 18:39:21,447:qiskit.aqua.operators.converters.circuit_sampler:DEBUG: Parameter conversion 15.61737 (ms)
2021-08-13 18:39:23,362:qiskit.aqua.algorithms.minimum_eigen_solvers.vqe:INFO: Energy evaluation returned [-32.60532084] - 192
9.98505 (ms), eval count: 99
2021-08-13 18:39:23,362:qiskit.aqua.operators.converters.circuit_sampler:DEBUG: Parameter conversion 0.00000 (ms)
2021-08-13 18:39:24,964:qiskit.aqua.algorithms.minimum_eigen_solvers.vqe:INFO: Energy evaluation returned [-32.34164714] - 160
2.39887 (ms), eval count: 100
2021-08-13 18:39:24,964:qiskit.aqua.algorithms.minimum_eigen_solvers.vqe:INFO: Optimization complete in 354.0857002735138 secon
ds.
Found opt_params [ 2.95777409  1.830187    1.48377791 -2.741093   -0.67009315  1.16726075
  1.45059764  3.04899493 -1.12283295  1.85708424  0.19077447  2.44121688
 -0.18375861 -1.50911504 -3.446049   -1.33315616 -1.72368652 -1.20728877
  0.15992551  1.44134823 -1.61017947  2.47188693 -3.17283326 -1.6206213
  4.05986751  3.07603628 -0.32736298  0.77813946  1.11263487 -1.32581648
 -0.12026996  0.96248825] in 100 evals
```

Figura 24: Ejecución del algoritmo VQE

En la figura anterior podemos ver que el algoritmo va cambiando los parámetros con el fin de obtener un nivel de energía mínimo. Si nos retrotraemos a la figura 21, podemos observar que habíamos configurado un máximo de 100 iteraciones. Esto no quiere decir que 100 sea el número óptimo de iteraciones que debamos poner. La forma correcta de determinar cuál es el número óptimo de iteraciones es ir probando aumentar paulatinamente el número de iteraciones hasta que lleguemos a una cantidad que ya se estabilice y ya no genere más beneficio por aumentar dichas iteraciones.

También hemos de añadir que, si no obtenemos buenos resultados cambiando el número de iteraciones, debemos centrarnos en cambiar de optimizador ya que cuanto más acertemos con este, menos iteraciones tendrá que hacer el procesador cuántico.

4.3 Resultados

En este apartado, analizaremos los diferentes resultados que ha ido calculando el algoritmo para ver si realmente nos compensa resolver nuestro problema mediante el paradigma cuántico. Probaremos distintas configuraciones para ver cual nos arroja mejores resultados. Hemos dividido este apartado en dos subapartados ya que vamos a probar el algoritmo tanto en el simulador como en el ordenador cuántico real.

4.3.1 Simulador QASM

En primer lugar, vamos a ejecutar el algoritmo clásico para ver el camino nos propone:

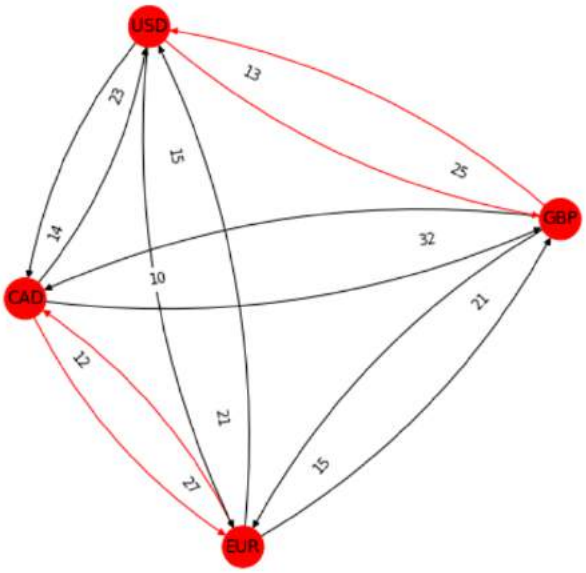

Figura 25: Grafo con los caminos alcanzados mediante el paradigma clásico

```
In [16]: ms = np.array(list(opt_result.variables_dict.values()))
         objective_value = opt_result.fval
         xs = np.zeros([length, length])
         xs[var_location] = ms#.get_values(var)
         path = list(zip(*np.nonzero(xs)))
         path = [(index2currency[i], index2currency[j]) for i, j in path]
         obj = np.exp(objective_value) - 1

         print('profit rate: {}, arbitrage path: {}'.format(obj, path))

         profit rate: 105298.99999999994, arbitrage path: [('GBP', 'USD'), ('EUR', 'CAD'), ('USD', 'GBP'), ('C
         AD', 'EUR')]
```

Figura 26: Código en Python para pintar la solución obtenida mediante el paradigma clásico

Esta es la solución que hemos alcanzado mediante el algoritmo clásico, también hemos de añadir que hemos necesitado un ordenador con una memoria bastante grande para poder ejecutar los cálculos sin problemas. Sospechamos que, si aumentamos la talla del problema, este crecerá demasiado para poder resolverse en un ordenador personal. Posteriormente aumentaremos la talla del problema

para comprobar si esta hipótesis se cumple. Ahora vamos a resolver el problema con la misma talla, pero con la ayuda del procesador cuántico.

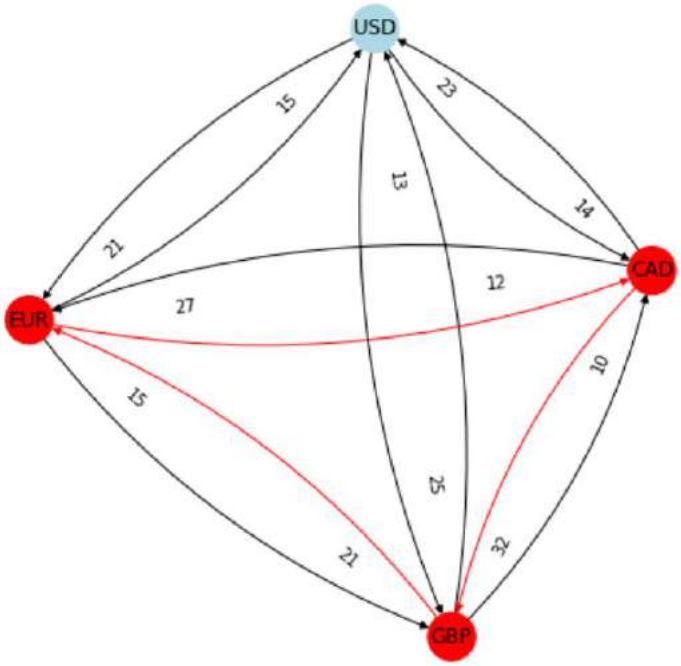

Figura 27: Grafo con los caminos alcanzados mediante el paradigma cuántico

```
profit rate: 104.49807999910892, arbitrage path: [('GBP', 'EUR'), ('EUR', 'CAD'), ('CAD', 'GBP')]
```

Figura 28: Salida de resultados alcanzados mediante el paradigma cuántico

Podemos ver que hay divergencia en la ejecución alcanzada de ambos paradigmas, obtenemos que para el paradigma cuántico no será necesario pasar por dólares para obtener el camino de máximo beneficio. Sospechamos que conforme incrementos el número de nodos, podremos incrementar el beneficio, ya que habrá distintos caminos para realizar los cambios, y por lo tanto más oportunidades de beneficio. Por lo tanto, para el siguiente caso vamos a añadir una nueva divisa: el franco suizo (CHF) que hará incrementar de complejidad el problema y así poder ver diferencias significativas.

```
~\anaconda3\lib\site-packages\qiskit\aqua\operators\primitive_ops\pauli_op.py in to_spmatrix(self)
    161             ValueError: invalid parameters.
    162         """
--> 163         return self.primitive.to_spmatrix() * self.coeff  # type: ignore
    164
    165     def __str__(self) -> str:

~\anaconda3\lib\site-packages\scipy\sparse\base.py in __mul__(self, other)
    473         if isscalarlike(other):
    474             # scalar value
--> 475             return self._mul_scalar(other)
    476
    477         if issparse(other):

~\anaconda3\lib\site-packages\scipy\sparse\data.py in _mul_scalar(self, other)
    122
    123     def _mul_scalar(self, other):
--> 124         return self._with_data(self.data * other)
    125
    126

~\anaconda3\lib\site-packages\scipy\sparse\compressed.py in _with_data(self, data, copy)
   1200         """
   1201         if copy:
-> 1202             return self.__class__((data, self.indices.copy(),
   1203                                    self.indptr.copy()),
   1204                                   shape=self.shape,

MemoryError: Unable to allocate 128. MiB for an array with shape (33554432,) and data type int32
```

Figura 29: Error de memoria insuficiente al intentar ejecutar el algoritmo clásico

Nombre	Estado	43% CPU	92% Memoria
Python		27,8%	11.530,6 ...

Figura 30: Uso de la memoria y CPU mientras ejecutamos el algoritmo

Podemos ver que, si tratamos de resolver el problema mediante el paradigma clásico, no tenemos memoria suficiente para realizar los cálculos. En la figura 29, nos salta un error en el cual nos dice que no ha podido cargar un array tan grande en la memoria. Se puede ver, en la figura 30, que mientras realizamos la ejecución estamos sobreexplotando la memoria del ordenador, y esto finalmente desemboca en el error que hemos explicado previamente.

Sin embargo, si ejecutamos el algoritmo con la ayuda del procesador cuántico no obtenemos error de memoria. Esto es debido a que la operación más costosa que es la multiplicación de la matriz por el vector, la realiza el procesador cuántico, y esto supone un ahorro de memoria muy significativo.

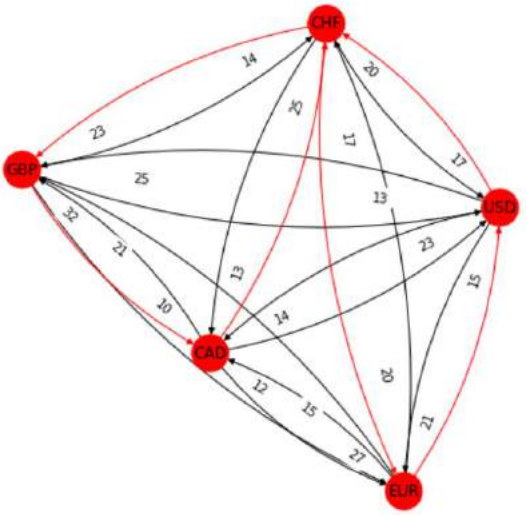

Figura 31: Grafo con los caminos alcanzados mediante el paradigma cuántico (talla = 5)

```
profit rate: 232458.17146844877, arbitrage path: [('GBP', 'CAD'), ('EUR', 'USD'), ('USD', 'CHF'), ('CAD', 'CHF'), ('CHF', 'GB
P'), ('CHF', 'EUR')]
```

Figura 32: Salida de resultados alcanzados mediante el paradigma cuántico (talla = 5)

Podemos ver en la figura 32 que logramos un beneficio muy superior al planteado previamente con 4 divisas solo. Ahora vamos a incrementar la talla a 6 divisas para ver cuáles son los resultados que alcanzamos. Sólo vamos a fijarnos en el enfoque cuántico ya que lógicamente nos va a saltar un error de memoria nuevamente cuando intentemos resolver el problema mediante el enfoque clásico. Esta vez vamos a incorporar el dólar australiano (AUD) a nuestro set de divisas.

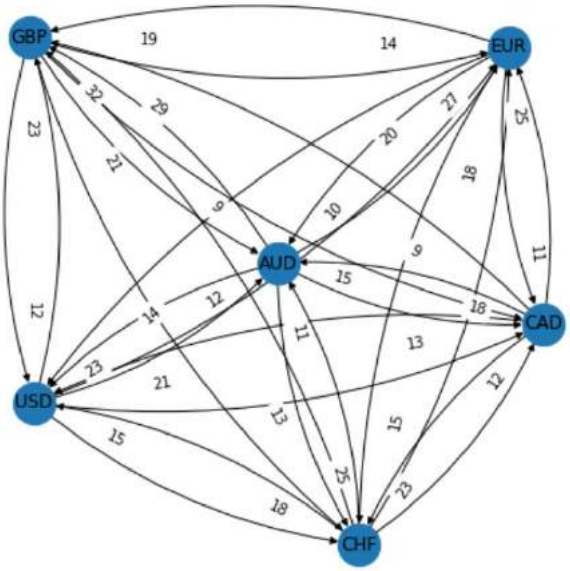

Figura 33: Grafo de precios de cambio (talla = 6)

```
~\anaconda3\lib\site-packages\qiskit\aqua\operators\converters\circuit_sampler.py in sample_circuits(self, circuit_sfns, param_
bindings)
    290             ready_circs = self._transpiled_circ_cache
    291
--> 292         results = self.quantum_instance.execute(ready_circs,
    293                                                 had_transpiled=self._transpile_before_bind)
    294

~\anaconda3\lib\site-packages\qiskit\aqua\quantum_instance.py in execute(self, circuits, had_transpiled)
    396
    397         else:
--> 398             result = run_qobj(qobj, self._backend, self._qjob_config,
    399                               self._backend_options, self._noise_config,
    400                               self._skip_qobj_validation, self._job_callback)

~\anaconda3\lib\site-packages\qiskit\aqua\utils\run_circuits.py in run_qobj(qobj, backend, qjob_config, backend_options, noise_
config, skip_qobj_validation, job_callback)
    332                     msg += ', ' + res.status
    333                     break
--> 334         raise AquaError('Circuit execution failed: {}'.format(msg))
    335
    336     return result

AquaError: 'Circuit execution failed: ERROR:  [Experiment 0] QasmSimulator: Insufficient memory for 36-qubit circuit using "sta
tevector" method. You could try using the "matrix_product_state" or "extended_stabilizer" method instead.'
```

Figura 34: Error de memoria insuficiente al intentar ejecutar el algoritmo cuántico

Podemos observar que, dada la complejidad del problema, este problema tampoco es resoluble mediante el procesador cuántico, ya que necesitamos más memoria. Hay que tener en cuenta que cada vez que añadimos otra divisa al problema, aumentamos considerablemente las necesidades de RAM. Esto es una limitación del simulador (por esto necesitamos ordenadores cuánticos reales). El ordenador cuántico no tiene RAM, el entrelazamiento es un proceso físico entre los qubits, y no necesitamos almacenar dos coeficientes complejos por cada pareja de qubits para calcular el efecto. En el siguiente apartado de este estudio, ejecutaremos el algoritmo en un ordenador cuántico real, pero sólo el de talla 5, ya que 36 qubits que es lo que necesita nuestra máquina si queremos ejecutar el problema con una talla superior, los tienen sólo máquinas bastante avanzadas que están fuera de nuestro alcance económico en este estudio.

Ahora, vamos a cambiar el optimizador clásico para el problema de talla 5. De esta forma veremos si logramos una mejoría en los resultados.

SLSQP:

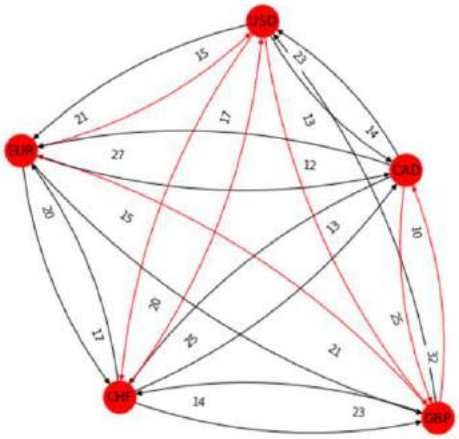

Figura 35: Grafo de precios (SLSQP)

```
profit rate: 186.21222014711762, arbitrage path: [('GBP', 'EUR'), ('GBP', 'CAD'), ('EUR', 'USD'), ('USD', 'GBP'), ('USD', 'CH
F'), ('CAD', 'GBP'), ('CHF', 'USD')]
```

Figura 36: Salida de resultados (SLSQP)

SPSA

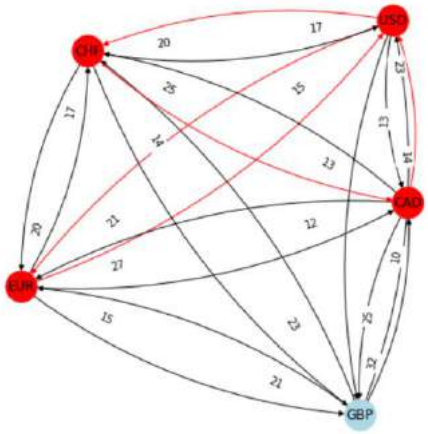

Figura 37: Grafo de precios (SPSA)

```
profit rate: -0.42377179887071137, arbitrage path: [('EUR', 'USD'), ('USD', 'EUR'), ('USD', 'CHF'),
('CAD', 'USD'), ('CHF', 'CAD')]
```

Figura 38: Salida de resultados (SPSA)

TNC

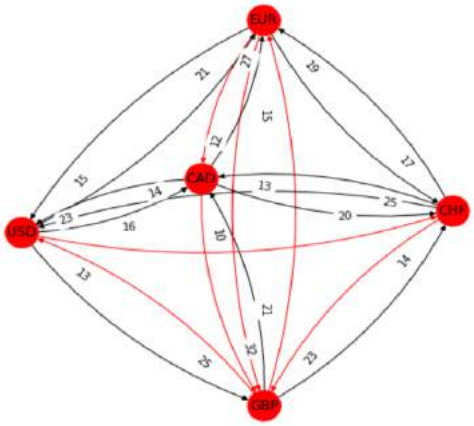

Figura 39: Grafo de precios (TNC)

```
profit rate: -0.999999996926992, arbitrage path: [('GBP', 'EUR'), ('GBP', 'USD'), ('EUR', 'GBP'), ('E
UR', 'CAD'), ('USD', 'CHF'), ('CAD', 'GBP'), ('CHF', 'GBP')]
```

Figura 40: Salida de resultados (TNC)

GSLS

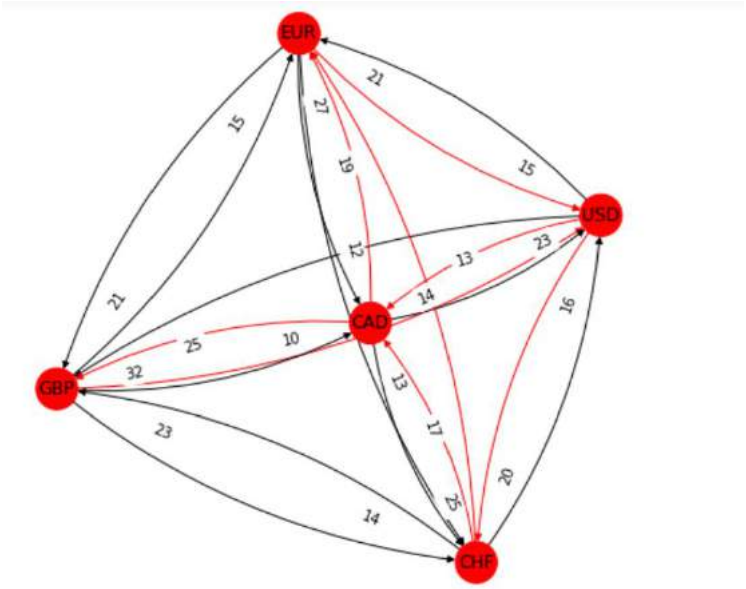

Figura 41: Grafo de precios (GSLS)

```
profit rate: -0.9989096425850164, arbitrage path: [('GBP', 'USD'), ('EUR', 'USD'), ('USD', 'CAD'),
('USD', 'CHF'), ('CAD', 'GBP'), ('CAD', 'EUR'), ('CHF', 'EUR'), ('CHF', 'CAD')]
```

Figura 42: Salida de resultados (GSLS)

NELDER_MEAD

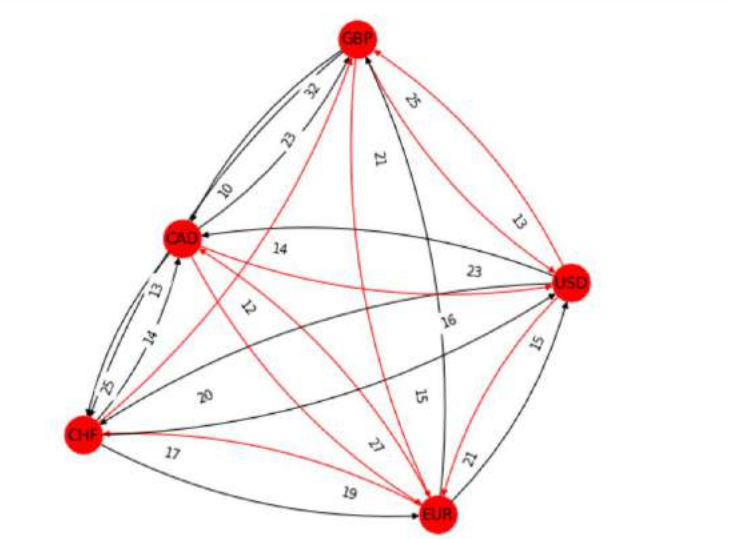

Figura 43: Grafo de precios (NELDER_MEAD)

```
profit rate: -0.9720868363814096, arbitrage path: [('GBP', 'EUR'), ('GBP', 'USD'), ('EUR', 'CAD'),
('EUR', 'CHF'), ('USD', 'GBP'), ('USD', 'EUR'), ('CAD', 'EUR'), ('CAD', 'USD'), ('CHF', 'GBP')]
```

Figura 44: Salida de resultados (NELDER_MEAD)

Después de probar con varias opciones, llegamos a la conclusión que el mejor optimizador para nuestro problema es COBYLA, ya que nos arroja un beneficio mayor. Hemos probado más optimizadores a parte de los que se han mostrado previamente, no obstante, estos no han llegado a converger y por lo tanto no los hemos tenido en cuenta.

4.3.2 IBM Quantum Lab

Quantum Lab es una herramienta de la empresa IBM que nos permite construir aplicaciones y experimentos cuánticos con Qiskit en un entorno basado en la nube. Con esta herramienta podemos escribir scripts que combinen código Qiskit, ecuaciones, visualizaciones y texto narrativo en el entorno Jupyter Notebook sin necesidad de instalar nada en nuestra computadora. Podemos ejecutar el código en simuladores o en hardware cuántico real.

Para acceder a esta herramienta, sólo necesitaremos un explorador web y una cuenta IBMid.
Visitaremos la url:
https://quantum-computing.ibm.com/ para iniciar sesión con nuestra cuenta IBM.

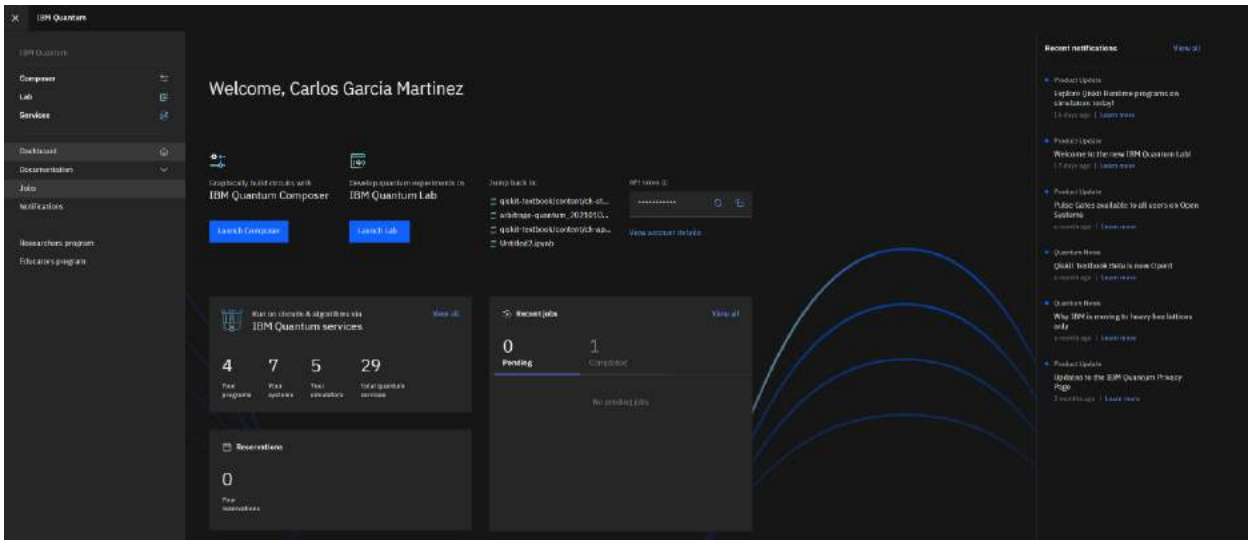

Figura 45: Pantalla de inicio de Quantum Lab

Nosotros hemos desarrollado el código de forma local, pero Quantum Lab nos da la opción de programar directamente en Jupyter Notebooks que ya tienen preinstaladas todas las librerías qiskit necesarias para la ejecución del código cuántico que queramos probar.

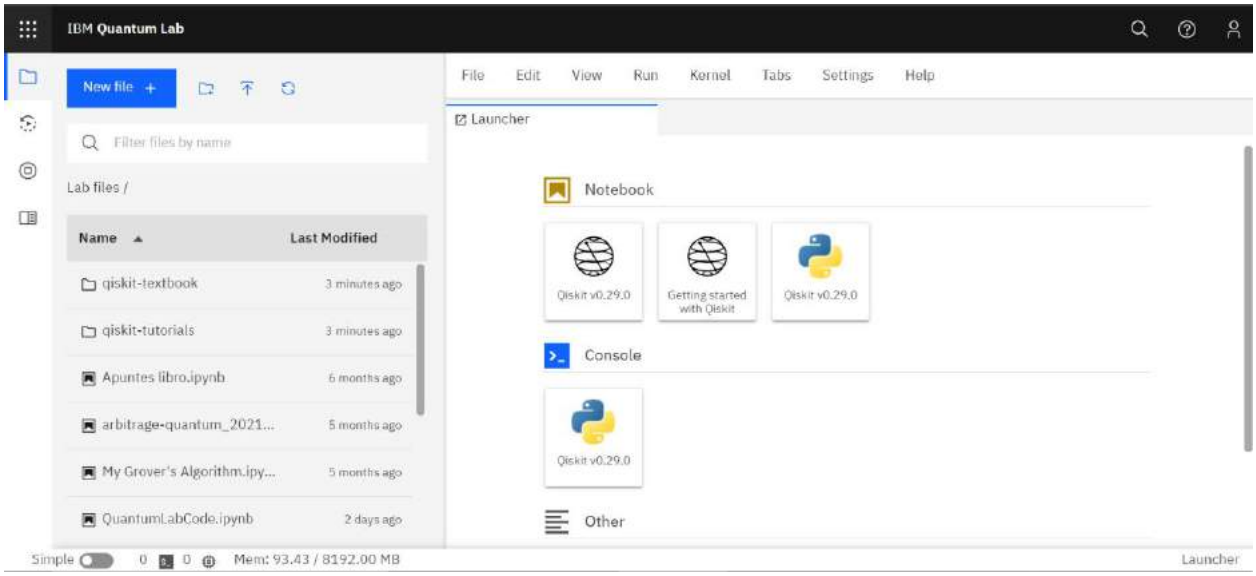

Figura 46: Pestaña Lab

Tal y como se muestra en la figura anterior, podemos ver todos los notebooks utilizados por nuestro usuario en el panel izquierdo. Nosotros tendremos que subir aquí el código que hemos desarrollado previamente. Podríamos haberlo desarrollado todo en Quantum Lab, no obstante, tenemos la certeza de que en nuestra máquina local funciona mejor, ya que nuestro algoritmo es de una complejidad considerable y por tanto una ejecución en la nube nos daría bastantes problemas. Por otra parte, si queremos ejecutar otro tipo de algoritmos que no necesiten tantos recursos, podemos utilizar únicamente Quantum Lab para todo el proceso.

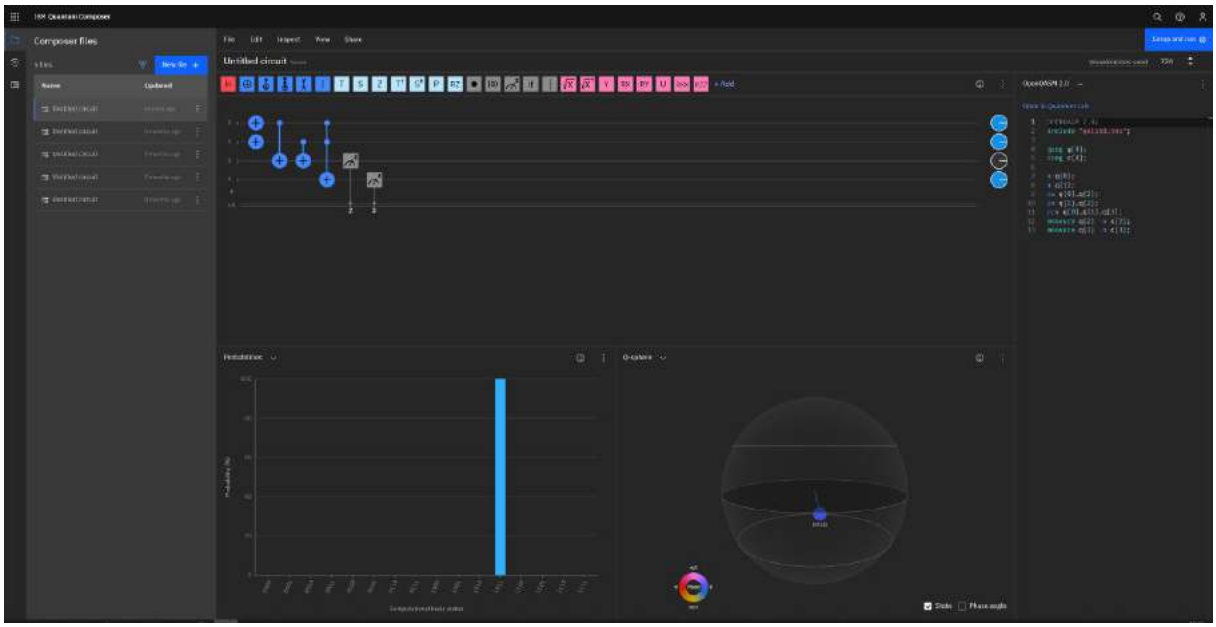
Figura 47: Composer en Quantum Lab

Hemos replicado el circuito cuántico de la figura 4 en Quantum Lab. Podemos ver que esta herramienta nos permite configurar circuitos cuánticos sin necesidad de conocer un lenguaje de programación. Podemos arrastrar las puertas que queramos en el diagrama y automáticamente la herramienta escribe el código Python correspondiente.

Además, tenemos información en tiempo real del histograma de resultados lo cual hace mucho más fácil el diseño del circuito en cuestión. A la derecha tenemos un diagrama donde nos indica la fase en

la cual encontramos los electrones. En este diagrama podemos conocer los grados de rotación que realiza el electrón sobre los ejes.

Finalmente, en la pestaña “Services” encontramos los distintos ordenadores cuánticos y su estado actual. Además, Quantum Lab nos ofrece distintos simuladores con los que también podemos ejecutar nuestros circuitos.

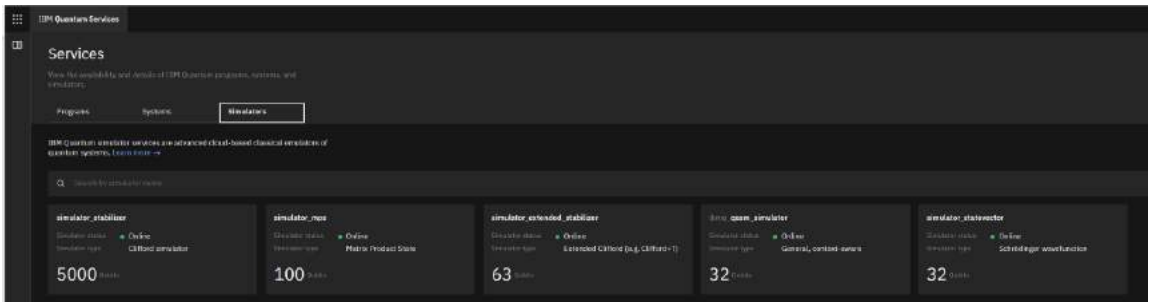

Figura 48: Simuladores en Quantum Lab

Ahora vamos a modificar nuestro código para ejecutar el algoritmo en un software cuántico real. Para ello debemos comprobar en el panel de computadores, cuales de ellos están libres y tienen la suficiente potencia para ejecutar nuestro código. Para realizar la ejecución en los ordenadores cuánticos reales debemos de preparar el código de nuestro algoritmo para que conecte a los servidores de IBM.

En primer lugar, iremos a la página de nuestra cuenta en Quantum Lab, y copiaremos el API token que utilizará nuestro código como autenticación para conectar a los servidores cuánticos.

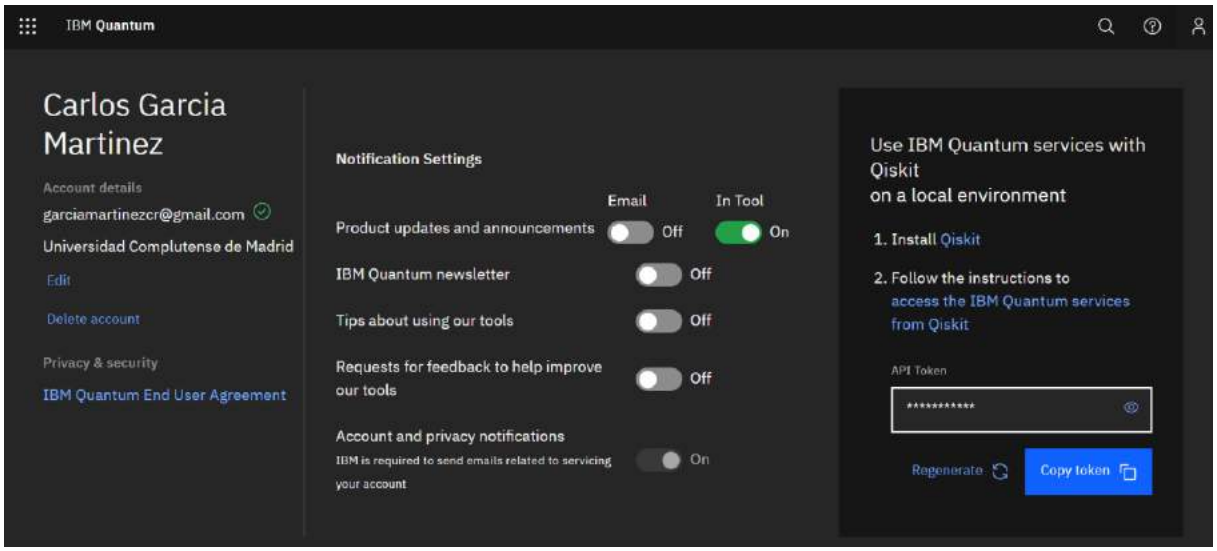

Figura 49: Perfil de Quantum Lab

Haremos clic en el botón azul “Copy token” para copiar la clave al portapapeles. Llamaremos al método *save_account* en el cual pasaremos el token como argumento. Posteriormente utilizaremos el método load_account para instanciar nuestra cuenta con el token que hemos obtenido.

En nuestro caso, sólo hará falta el método load_account ya que dado que al haber subido nuestra notebook a Quantum Lab, ya tenemos precargadas nuestras credenciales.

```
[26]: # set classical optimizer
      maxiter = 100
      optimizer = COBYLA(maxiter=maxiter)

      # set variational ansatz
      var_form = RealAmplitudes(length, reps=1)
      m = var_form.num_parameters

      # set backend
      #backend_name = 'qasm_simulator'  # use this for QASM simulator
      #backend_name = 'statevector_simulator'  # use this for statevector simlator
      #Real backend

      IBMQ.load_account()


      __init__.discover_credentials:INFO:2021-08-20 09:55:48,261: Using credentials from qiskitrc

[26]: <AccountProvider for IBMQ(hub='ibm-q', group='open', project='main')>
```

Figura 50: Ejecución del código que carga la cuenta

Tal y como se ve en la figura anterior, nos está diciendo que está utilizando las credenciales de la cuenta en la cual hemos iniciado sesión. Ahora vamos a seleccionar el computador cuántico sobre el cual vamos a ejecutar nuestro programa, en primer lugar lo que vamos a hacer es listar aquellos que podemos utilizar. Nosotros tenemos una cuenta gratuita por lo que sólo podremos acceder a las máquinas menos potentes. Los computadores con mayor potencia están reservados para usuarios con suscripción de pago.

```
[29]: provider = IBMQ.get_provider(group='open')
      provider.backends()

[29]: [<IBMQSimulator('ibmq_qasm_simulator') from IBMQ(hub='ibm-q', group='open', project='main')>,
       <IBMQBackend('ibmq_armonk') from IBMQ(hub='ibm-q', group='open', project='main')>,
       <IBMQBackend('ibmq_santiago') from IBMQ(hub='ibm-q', group='open', project='main')>,
       <IBMQBackend('ibmq_bogota') from IBMQ(hub='ibm-q', group='open', project='main')>,
       <IBMQBackend('ibmq_lima') from IBMQ(hub='ibm-q', group='open', project='main')>,
       <IBMQBackend('ibmq_belem') from IBMQ(hub='ibm-q', group='open', project='main')>,
       <IBMQBackend('ibmq_quito') from IBMQ(hub='ibm-q', group='open', project='main')>,
       <IBMQSimulator('simulator_statevector') from IBMQ(hub='ibm-q', group='open', project='main')>,
       <IBMQSimulator('simulator_mps') from IBMQ(hub='ibm-q', group='open', project='main')>,
       <IBMQSimulator('simulator_extended_stabilizer') from IBMQ(hub='ibm-q', group='open', project='main')>,
       <IBMQSimulator('simulator_stabilizer') from IBMQ(hub='ibm-q', group='open', project='main')>,
       <IBMQBackend('ibmq_manila') from IBMQ(hub='ibm-q', group='open', project='main')>]
```

Figura 51: Lista de computadores cuánticos disponibles

Aunque tengamos disponibles todos los computadores que se muestran en la figura anterior, tenemos que seleccionar el que mejor se adapta a nuestro problema. Realizando distintas pruebas, hemos llegado a la conclusión de que nuestro programa tendrá las necesidades que se detallan a continuación.

NÚMERO DE DIVISAS	NÚMERO DE QÚBITS
2	5
3	9
4	16
5	25

Tabla 5: Necesidades de procesador cuántico

Teniendo en cuenta la tabla anterior, sólo podremos ejecutar una instancia del problema que tenga dos divisas en hardware real. Esto es debido a que los ordenadores que tenemos a nuestra disposición no tienen más de 7 qubits. Por otra parte, no sólo hemos de fijarnos en los qubits y el volumen cuántico que tienen los ordenadores a la hora de ejecutar un programa, también hemos de darnos cuenta de la arquitectura que tienen.

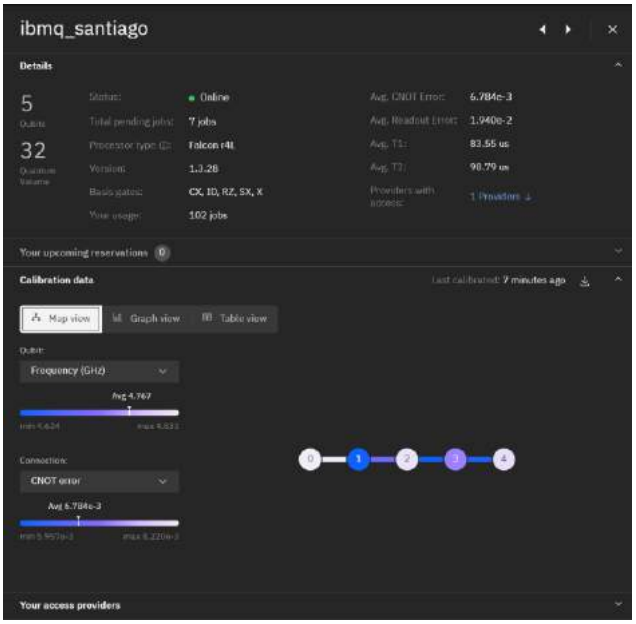

Figura 52: Arquitectura de ibmq_santiago

Como podemos ver en la figura, los 5 qubits que componen el procesador están conectados dos a dos entre sí. Por ejemplo, están conectados el 0 y el 1 pero no el 0 y el 4. Esto puede afectarnos a la hora de diseñar circuitos, ya que nos limitará de una forma u otra al configurar las puertas entre los diferentes qubits Por otra parte, existen otros computadores más complejos como el de la figura 53 que tiene 65 qúbits pero no tenemos acceso a este ordenador para poder ejecutar nuestro programa.

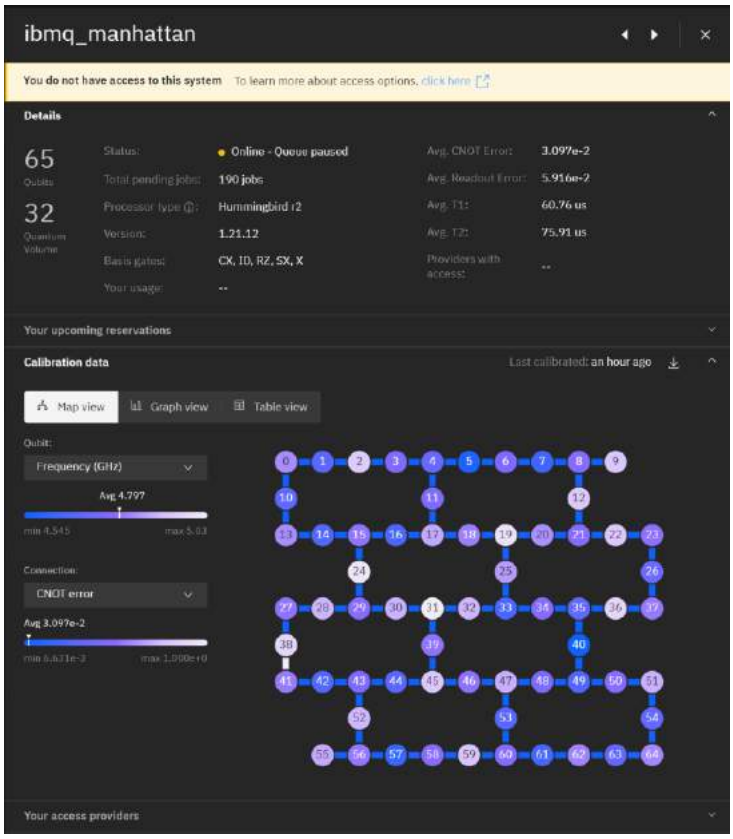

Figura 53: Arquitectura de ibm_q Manhattan

En este ordenador podemos ver los qubits que están conectados entre sí, y que también nos limitan al realizar ejecuciones. Por este motivo, la arquitectura es un factor determinante que deberemos tener en cuenta a la hora de programar circuitos cuánticos.

Dicho esto, vamos a ejecutar nuestro algoritmo en el ordenador ibm_q Santiago. Al tener sólo dos divisas, el hecho de ejecutarlo va a tener solo un objetivo ilustrativo, para que el lector compruebe que nuestro algoritmo está preparado para hardware cuántico real. Si quisiéramos ver la utilidad práctica del algoritmo en un hardware cuántico real, necesitaríamos pagar por un ordenador con especificaciones superiores, como por ejemplo el que hemos mostrado en la figura 53.

Figura 54: Log que muestra el envío a la cola de ibmq_santiago

En el log expuesto previamente podemos ver que nuestro programa se envía a una cola de trabajos que tiene el ordenador. Una vez que este llega a la primera posición, se ejecuta en el ordenador y después se recupera un objeto de tipo resultado, de esta forma hace las distintas iteraciones y finalmente nos muestra los parámetros calculados.

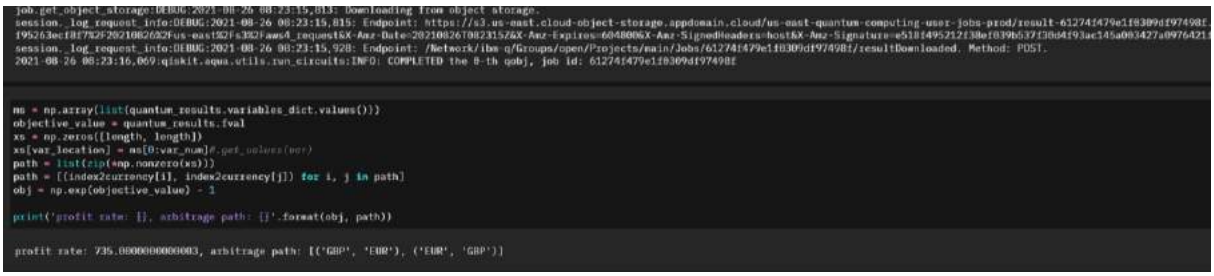

Figura 55: Resultados de la ejecución real

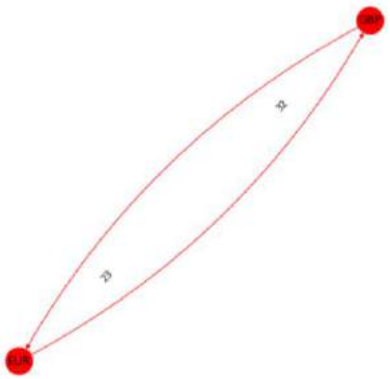

Figura 56: Grafo con el resultado de la ejecución real

Lógicamente, nuestro programa sólo ha encontrado el único camino que había disponible en el grafo. No obstante, tal y como hemos explicado previamente esta ejecución sólo era para demostrar que nuestro algoritmo se puede probar en ordenadores cuánticos reales, que en última instancia es donde se ejecutarían si en un futuro algún tipo de institución o empresa quiere generar beneficio con este método.

5. Conclusiones

En este apartado, nuestro objetivo va a ser determinar si hemos tenido éxito en nuestro trabajo e intentar predecir cómo afectará en un futuro esta aportación. Para ello, repasaremos con detalle todos los hitos que hemos ido consiguiendo y evaluaremos si dichos hitos cumplen los objetivos previamente marcados en el proyecto. Posteriormente haremos un análisis prospectivo en el que esbozaremos un horizonte temporal próximo donde se reflexionará acerca de nuestra aportación.

Primeramente, en los meses que van de diciembre a enero nos introducimos en el mundo de la computación cuántica y descubrimos sus características más relevantes. Además, la comparamos con la computación clásica para ver qué ventajas tenía y por qué se le espera un futuro tan brillante.

Más adelante, aproximadamente de marzo a abril estuvimos practicando circuitos cuánticos básicos para conocer de una forma práctica cómo se programaban este tipo de entidades. También profundizamos en algoritmos cuánticos más complejos como el algoritmo de Grover o VQE. Con ello vimos las posibilidades que podía ofrecer en nuestro contexto del arbitraje.

Durante el mes de junio, empezamos a elaborar las restricciones que conformarían la base de nuestro problema. Además, adaptamos VQE a nuestro contexto ya que VQE se diseñó en un principio para simular el comportamiento de las moléculas en reacciones químicas.

Finalmente, en los meses de julio y agosto se probaron varias ejecuciones para ver cómo respondía el algoritmo y así ver en qué situaciones se comportaba mejor. Al principio nos dimos cuenta de que el algoritmo era muy lento, tanto en el paradigma cuántico como en el paradigma clásico, por este motivo tuvimos la idea de transformar los números a enteros. El hecho de transformar los números de la matriz de coma flotante a enteros nos supuso una optimización muy significativa ya que pasamos de usar números en coma flotante de 64 bits a números enteros de 32 bits. Esto llevó a una reducción del tiempo de ejecución a la mitad y un uso de memoria bastante más reducido.

Nos dimos cuenta de que para tallas superiores a 4, no podemos resolver el problema mediante el paradigma clásico ya que este provoca un desborde en el buffer de memoria, generando un vector que es demasiado grande para ser almacenado debido a sus dimensiones. No obstante, pudimos ver que para la misma talla era resoluble a través del paradigma cuántico. Gracias a esto, pudimos sacar un beneficio sustancialmente superior al que obteníamos con la talla inferior. Esto es una consecuencia directa de la propiedad llamada superposición cuántica que introdujimos en el primer capítulo y consiste en que un qubit a diferencia del bit tiene 3 estados: 0, 1 y la superposición de ambos. De tal forma que por cada qubit más que tiene nuestra máquina, podemos almacenar dos unidades más de información, con el bit sólo podemos almacenar una unidad más.

Llegados a este punto, ya habíamos cumplido con los tres objetivos que habíamos planteado en un principio, no obstante, nos propusimos hacer algo que estaba fuera del alcance del proyecto inicial, esto era probar nuestro algoritmo en un hardware cuántico real. Las ejecuciones se hacían en un simulador ya que este aparte de tener más potencia, las ejecuciones duraban menos tiempo. Hemos de pensar que al estar en una interfaz web global, muchos usuarios pueden usarlo al mismo tiempo que nosotros, esto puede llevar a tiempos de espera largos. En el registro (fig. 54) puede verse que cada iteración que hace el algoritmo se envía a una cola donde espera su turno hasta que le toca y puede ejecutarse. Después el ordenador devuelve un objeto con el resultado de la iteración.

Cuando nos pusimos a ejecutar en el ordenador cuántico real, nos dimos cuenta de que nuestro problema era demasiado grande como para ejecutarlo con los ordenadores gratuitos. Por lo tanto, la solución fue reducir la talla del problema hasta tal punto que fuera ejecutable por uno de los

ordenadores gratuitos. No obstante, la talla era demasiado pequeña como para encontrar un beneficio real.

En última instancia, este trabajo se ha llevado a cabo desde un principio con perspectivas de futuro. Es decir, éramos conscientes desde un primer momento que el hardware cuántico actual no nos permitiría obtener beneficio ya que en estos momentos un ordenador cuántico no supone una gran ventaja respecto a un ordenador normal, pero esto es debido a que la tecnología cuántica no está desarrollada.

Según la ley de Moore, se duplica la cantidad de transistores en un procesador, pero esto sólo es aplicable en el paradigma clásico, cuando lo intentamos extrapolar al paradigma cuántico, obtenemos la ley de Neven. Esta se formuló en el interior del laboratorio de inteligencia artificial cuántica de Google, y dice que los computadores cuánticos están ganando poder computacional a una tasa doblemente exponencial (Undurraga, 2019).

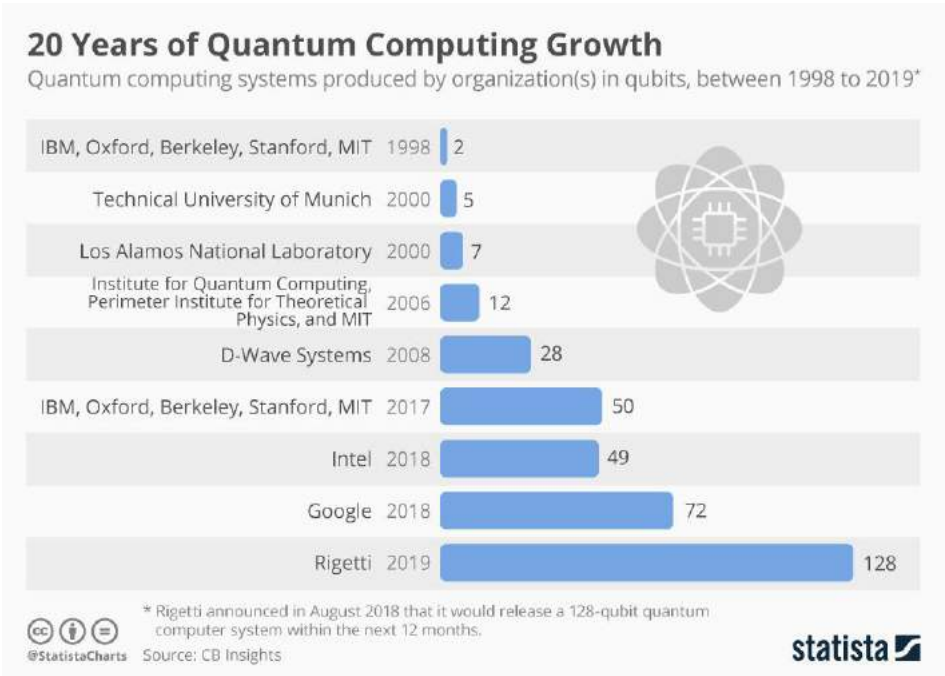

Figura 57: Evolución de los computadores cuánticos

Viendo estos datos, podemos decir que nuestro proyecto será viable dentro de unos 5 años, cuando los ordenadores cuánticos tengan la potencia suficiente para crear grafos con tallas suficientemente grandes. Según IBM, en 2023 tendrá operativo un ordenador cuántico de 1121 qubits, lo que supone multiplicar por 17 la capacidad actual de 65 bits (Hernández, 2020).

La computación cuántica todavía está en sus fases iniciales, y todavía estamos lejos de mostrar todo el potencial que tiene, no obstante, todos los investigadores, ingenieros, estudiantes… que estamos contribuyendo de una manera u otra a que la tecnología avance, aportamos en cierta manera un poco de luz en este universo tan maravillosamente complejo.

6. Bibliografía

Alberto Peruzzo, J. M. (23 de julio de 2014). *Nature communications*. Obtenido de Nature communications: https://www.nature.com/articles/ncomms5213

Arias, A. S. (18 de julio de 2017). *Economipedia.* Obtenido de Economipedia: https://economipedia.com/definiciones/forward.html

arXiv:1912.04088 [quant-ph] Cornell University. (6 de junio de 2021). Obtenido de arXiv:1912.04088 [quant-ph] Cornell University: https://arxiv.org/abs/1912.04088

Aspuru-Guzik, A. (9 de septiembre de 2005). *Science*. Obtenido de Science: https://science.sciencemag.org/content/309/5741/1704

BBVA. (2019). Obtenido de https://www.bbva.com/es/que-es-la-supremacia-cuantica/

Braun, M. C. (25 de junio de 2018). *C Braun*. Obtenido de https://medium.com/@markus.c.braun/a-brief-history-of-quantum-computing-a5babea5d0bd

Cambridge University Press. (11 de Mayo de 2021). Obtenido de Cambridge University Press: https://www.cambridge.org/core/journals/financial-history-review/article/origins-of-arbitrage/FD1A30E926696989376EDF449F9797BF

Efinance Management. (26 de junio de 2020). Obtenido de Efinance Management: https://efinancemanagement.com/derivatives/currency-arbitrage

Hernández, I. (18 de septiembre de 2020). *Business Insider*. Obtenido de https://www.businessinsider.es/ibm-lanzara-2023-ordenador-cuantico-1121-qubits-719881

Investopedia. (12 de abril de 2021). Obtenido de Investopedia: https://www.investopedia.com/terms/c/cash-and-carry-arbitrage.asp

Lanyon, B. (10 de enero de 2010). *Nature chemistry*. Obtenido de Nature chemistry: https://www.nature.com/articles/nchem.483

Lewis, M. (27 de mayo de 2017). *Missouri Western State University*. Obtenido de Missouri Western State University: https://arxiv.org/ftp/arxiv/papers/1705/1705.09844.pdf

Microsoft. (2 de enero de 2021). Obtenido de https://docs.microsoft.com/en-us/azure/quantum/concepts-the-qubit

N Cody Jones, J. D. (27 de noviembre de 2012). *Institute of Physics.* Obtenido de Institute of Physics: https://iopscience.iop.org/article/10.1088/1367-2630/14/11/115023

Probasco, J. (6 de junio de 2021). *Investopedia*. Obtenido de Investopedia: https://www.investopedia.com/currency-conversion-fee-definition-4768870

Qiskit Textbook. (11 de Julio de 2021). Obtenido de Qiskit Textbook: https://qiskit.org/textbook/ch-states/single-qubit-gates.html

Sharma, S. (5 de junio de 2020). *Towards Data Science*. Obtenido de https://towardsdatascience.com/the-ultimate-beginners-guide-to-quantum-computing-and-its-applications-5b43c8fbcd8f

The quantum daily. (26 de mayo de 2020). Obtenido de https://www.thequantumdaily.com/2020/05/26/tqd-exclusive-the-history-of-quantum-computing/

The Stanford Encyclopedia of Philosophy. (2019). Obtenido de https://plato.stanford.edu/archives/win2019/entries/qt-quantcomp

Undurraga, C. (9 de septiembre de 2019). *Medium*. Obtenido de https://medium.com/option-blog/superando-la-ley-de-moore-con-la-computaci%C3%B3n-cu%C3%A1ntica-3ca0b87a6590

Vasconcelos, F. (diciembre de 2019). *Massachusetts Institute of Technology.* Obtenido de http://web.mit.edu/francisc/www/MURJ_Full_Article.pdf

7. Anexo: Código Python desarrollado

Find the most profitable arbitrage opportunity with Quantum Computers

Given a market condition as a directed graph, find a closed cycle in the graph that represents the most profitable arbitrage opportunity. We solve this problem with Qiskit.

Here are some market conditions:

In [ ]:
```
!pip install forex_python
```

In [ ]:
```
from forex_python.converter import CurrencyRates
import pandas as pd

currency_set = ['GBP', 'EUR', 'USD', 'CAD', 'CHF']

c = CurrencyRates()

rates = []


for currency_index in range(len(currency_set)):
    temp_list = []
    for second_index in range(len(currency_set)):
        temp_list.append(c.get_rate(currency_set[currency_index], currency_set[second_index])
                         if currency_index!=second_index else 0)
    rates.append(temp_list)

df = (pd.DataFrame.from_records(rates)).transpose()
df.columns = currency_set
df.set_index([pd.Index(currency_set)], inplace=True)
```

In [ ]:
```
df.dtypes
##Transformation of the df to integers
old_df = df.copy()
##Scale to the maximum
maximum = 0
qubits = 5
for index, row in df.iterrows(): #Cuidado, complejidad n^2
    for x in currency_set:
        maximum = max(maximum,float(row[x]))

print(maximum)
###Divide by the maximum
for index, row in df.iterrows():
    for x in currency_set:
        row[x] = (float(row[x]) / float(maximum))
        row[x] = int(row[x] * 2**qubits)
        row[x] = row[x].astype(int)
#Change df types
df = df.astype('int')
```

In [ ]:
```
transit_price_matrix = df.values
print(transit_price_matrix)
```

https://stackoverflow.com/questions/60262159/generating-directed-graph-with-parallel-labelled-edges-vertices-in-python

In [ ]:
```
from qiskit.circuit.library import RealAmplitudes
from qiskit.aqua.components.optimizers import COBYLA, SLSQP, SPSA, CG, GSLS, NELDER_MEAD,POWELL
from qiskit.aqua.algorithms import NumPyMinimumEigensolver, VQE
from qiskit.aqua.operators import PauliExpectation, CVaRExpectation
from qiskit.optimization import QuadraticProgram
from qiskit.optimization.converters import LinearEqualityToPenalty
from qiskit.optimization.algorithms import MinimumEigenOptimizer
from qiskit import execute, Aer
from qiskit.aqua import aqua_globals

import numpy as np
import matplotlib.pyplot as plt
from docplex.mp.model import Model
```

In [ ]:
```
import logging
logger = logging.getLogger()
logger.setLevel(logging.DEBUG)
# setup aqua logging
import logging
from qiskit.aqua import set_qiskit_aqua_logging
set_qiskit_aqua_logging(logging.DEBUG) # choose INFO, DEBUG to see the log
```

Definition of the problem:

$$\max_{x \in \{0,1\}^{n \times n}} x\Sigma$$

where we use the following notation:

- $x \in \{0,1\}^{n \times n}$ denotes the matrix of binary decision variables, which indicate which edge to pick ($x[i] = 1$) and which not to pick ($x[i] = 0$).
- $\Sigma \in R^{n \times n}$ specifies the change rate between the currencies(transit_price_matrix).
- and L denotes the max path length, i.e. the number of exchanges to be performed to close the cicle.

We assume the following simplifications:
all assets have the same intrinsic value (normalized to 1)

In [ ]:
```
length = len(currency_set)
currency2index = {item: i for i, item in enumerate(currency_set)}
index2currency = {val: key for key, val in currency2index.items()}
```

Locate all the trading-feasible pairs so that decision variables can be located, used to reduced the number of decision variables, to accelerate the modelling speed.

In [ ]:
```
var_location = np.zeros([length, length])

for from_cur, to_cur in list(G.edges()):
    from_index = currency2index[from_cur]
    to_index = currency2index[to_cur]
    var_location[from_index, to_index] = 1
    var_location[to_index, from_index] = 1

var_location = var_location == 1

print(var_location)
```

Create docplex model

In [ ]:
```
mdl = Model('arbitrage_matrix')

var_num = int(np.sum(var_location))

var = mdl.binary_var_list(var_num, name='x')

x = np.zeros([length, length])
x = x.astype('object')
x[var_location] = var
```

Set optimization constraints for the model

In [ ]:
```
# 1. closed-circle arbitrage requirement, for each currency, transit-in equals transit-out.
mdl.add_constraints(
    mdl.sum(x[currency, :]) == mdl.sum(x[:, currency]) for currency in range(length))

# 2. each currency can only be transited-in at most once.
mdl.add_constraints(mdl.sum(x[currency, :]) <= 1 for currency in range(length))

# 3. the whole arbitrage path should be less than a given length
path_length = None
if isinstance(path_length, int):
    mdl.add_constraint(mdl.sum(x) <= path_length)
```

Objective

In [ ]:
```
final_transit_matrix = np.log(transit_price_matrix)
final_transit = final_transit_matrix[var_location]
final_x = x[var_location]

objective = mdl.sum(final_x * final_transit)

mdl.maximize(objective)
```

In [ ]:
```
# case to
qp = QuadraticProgram()
qp.from_docplex(mdl)
```

In [ ]:
```
print(qp.export_as_lp_string())
```

Solve clasically for comparison

it takes several minutes to optimize... be patient

In [ ]:
```
'''
ValueError: 'to_matrix' will return an exponentially large vector,
in this case '16777216' elements. Set aqua_globals.massive=True
or the method argument massive=True if you want to proceed.
'''
aqua_globals.massive=True

# solve classically as reference
opt_result = MinimumEigenOptimizer(NumPyMinimumEigensolver()).solve(qp)
opt_result
```

In [ ]:
```
ms = np.array(list(opt_result.variables_dict.values()))
objective_value = opt_result.fval
xs = np.zeros([length, length])
xs[var_location] = ms#.get_values(var)
path = list(zip(*np.nonzero(xs)))
path = [(index2currency[i], index2currency[j]) for i, j in path]
obj = np.exp(objective_value) - 1

print('profit rate: {}, arbitrage path: {}'.format(obj, path))
```

In [ ]:
```
# Set the figure size
plt.figure(figsize=(8,8))

# Get the edge labels and round them to 3 decimal places
# for more clarity while drawing
edge_labels = dict([((u,v), round(d['weight'], 3))
                    for u,v,d in G.edges(data=True)])

# Get the layout
pos = nx.spring_layout(G)

# Set different colors for edges and nodes in path
node_colors = ['red' if node in (node for edge in path for node in edge) else 'lightblue'
               for node in G.nodes()]
edge_colors = ['red' if edge in path else 'black'  for edge in G.edges()]

# Draw edge labels and graph
nx.draw_networkx_edge_labels(G,pos,edge_labels=edge_labels,
                             label_pos=0.25, font_size=10)
nx.draw(G, pos, with_labels=True,
        node_color=node_colors, edge_color=edge_colors,
        connectionstyle='arc3, rad = 0.15',
        node_size=800)

plt.show()
```

Use several converters to adapt the problem for Quantum algorithms

Converter: Eliminate inequalities

In [ ]:
```
from qiskit.optimization.converters import InequalityToEquality

conv = InequalityToEquality()
qp1 = conv.convert(qp)
```

In [ ]:
```
print(qp1.export_as_lp_string())
```

Converter: All variables to binary

In [ ]:
```
from qiskit.optimization.converters import IntegerToBinary
conv2 = IntegerToBinary()
qp2 = conv2.convert(qp1)
```

In [ ]:
```
print(qp2.export_as_lp_string())
```

Converter: Convert the problem to an unconstrained problem, with a penalty as the problem length

In [ ]:
```
conv3 = LinearEqualityToPenalty(penalty=length)
qp3 = conv3.convert(qp2)
```

In [ ]:
```
print(qp3.export_as_lp_string())
```

Solve again classically as reference

In [ ]:
```
opt_result = MinimumEigenOptimizer(NumPyMinimumEigensolver()).solve(qp3)
opt_result
```

In [ ]:
```
ms = np.array(list(opt_result.variables_dict.values()))
objective_value = opt_result.fval
xs = np.zeros([length, length])
xs[var_location] = ms[0:var_num]#.get_values(var)
path = list(zip(*np.nonzero(xs)))
path = [(index2currency[i], index2currency[j]) for i, j in path]
obj = np.exp(objective_value) - 1

print('profit rate: {}, arbitrage path: {}'.format(obj, path))
```

In [ ]:
```
# Set the figure size
plt.figure(figsize=(8,8))

# Get the edge labels and round them to 3 decimal places
# for more clarity while drawing
edge_labels = dict([((u,v), round(d['weight'], 3))
                    for u,v,d in G.edges(data=True)])

# Get the layout
pos = nx.spring_layout(G)

# Set different colors for edges and nodes in path
node_colors = ['red' if node in [node for edge in path for node in edge] else 'lightblue'
               for node in G.nodes()]
edge_colors = ['red' if edge in path else 'black'  for edge in G.edges()]

# Draw edge labels and graph
nx.draw_networkx_edge_labels(G,pos,edge_labels=edge_labels,
                             label_pos=0.25, font_size=10)
nx.draw(G, pos, with_labels=True,
        node_color=node_colors, edge_color=edge_colors,
        connectionstyle='arc3, rad = 0.15',
        node_size=800)

plt.show()
```

Solve with Quantum Computing

In [ ]:
```
_ising, offset = qp3.to_ising()
```

In [ ]:
```
print(_ising)
```

Minimum Eigen Optimizer using VQE

In [ ]:
```
# set classical optimizer
maxiter = 100
optimizer = COBYLA(maxiter=maxiter)

# set variational ansatz
var_form = RealAmplitudes(length, reps=1)
m = var_form.num_parameters

# set backend
backend_name = 'qasm_simulator'  # use this for QASM simulator
#backend_name = 'statevector_simulator' # use this for statevector simlator

backend = Aer.get_backend(backend_name)
```

In [ ]:
```
# initialize VQE
vqe = VQE(optimizer=optimizer, var_form=var_form, quantum_instance=backend)

# initialize optimization algorithm based on CVaR-VQE
opt_alg = MinimumEigenOptimizer(vqe)
```

In [ ]:
```
# solve problem
quantum_results = opt_alg.solve(qp3)
```

In [ ]:
```
ms = np.array(list(quantum_results.variables_dict.values()))
objective_value = quantum_results.fval
xs = np.zeros([length, length])
xs[var_location] = ms[0:var_num]#.get_values(var)
path = list(zip(*np.nonzero(xs)))
path = [(index2currency[i], index2currency[j]) for i, j in path]
obj = np.exp(objective_value) - 1

print('profit rate: {}, arbitrage path: {}'.format(obj, path))
```

In [ ]:
```
# Set the figure size
plt.figure(figsize=(8,8))

# Get the edge labels and round them to 3 decimal places
# for more clarity while drawing
edge_labels = dict([((u,v), round(d['weight'], 3))
                    for u,v,d in G.edges(data=True)])

# Get the layout
pos = nx.spring_layout(G)

# Set different colors for edges and nodes in path
node_colors = ['red' if node in [node for edge in path for node in edge] else 'lightblue'
               for node in G.nodes()]
edge_colors = ['red' if edge in path else 'black'  for edge in G.edges()]

# Draw edge labels and graph
nx.draw_networkx_edge_labels(G,pos,edge_labels=edge_labels,
                             label_pos=0.25, font_size=10)
nx.draw(G, pos, with_labels=True,
        node_color=node_colors, edge_color=edge_colors,
        connectionstyle='arc3, rad = 0.15',
        node_size=800)

plt.show()
```

In [ ]:

```
import functools
import itertools
import numpy
import operator
import perfplot


def forfor(a):
    return [item for sublist in a for item in sublist]


def sum_brackets(a):
    return sum(a, [])


def functools_reduce(a):
    return functools.reduce(operator.concat, a)


def functools_reduce_iconcat(a):
    return functools.reduce(operator.iconcat, a, [])


def itertools_chain(a):
    return list(itertools.chain.from_iterable(a))


def numpy_flat(a):
    return list(numpy.array(a).flat)


def numpy_concatenate(a):
    return list(numpy.concatenate(a))


perfplot.show(
    setup=lambda n: [list(range(10))] * n,
    # setup=lambda n: [list(range(n))] * 10,
    kernels=[
        forfor,
        sum_brackets,
        functools_reduce,
        functools_reduce_iconcat,
        itertools_chain,
        numpy_flat,
        numpy_concatenate,
    ],
    n_range=[2 ** k for k in range(16)],
    xlabel="num lists (of length 10)",
    # xlabel="len lists (10 lists total)"
)
```