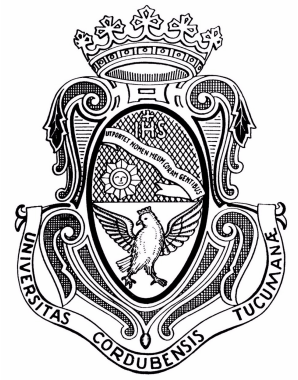


Facultad de Matemática,
Astronomía, Física y Computación

Trabajo Especial de Licenciatura en Física

Implementación numérica del algoritmo de factorización de Shor

Algoritmos eficientes para la Exponenciación Modular, Paralelización de Compuertas, Escalabilidad y Errores.

Santiago Emilio Bogino

Directores:
Dr. Omar Osenda Dr. Nicolás Wolovick

Resumen - Abstract

A pesar del descubrimiento de muchos algoritmos cuánticos con gran potencial teórico, a la fecha la principal limitación de la Computación Cuántica en la práctica es la poca escalabilidad de las computadoras cuánticas. El algoritmo de factorización de Shor[1] es uno de estos novedosos algoritmos, y permite una mejora exponencial en el costo computacional de la factorización de enteros grandes. La complejidad de la factorización utilizando computadoras clásicas es la base de la fortaleza de muchos de los sistemas criptográficos de clave pública más utilizados en la actualidad, y una potencial implementación escalable del algoritmo de Shor los volvería obsoletos.
En este trabajo se presenta y desarrolla la teoría para la simulación de algoritmos cuánticos en computadoras clásicas. A partir de ella, se lleva a cabo una implementación numérica para la simulación de compuertas y circuitos cuánticos, introduciendo técnicas algebraicas para reducir el número de operaciones, a la vez que se presentan versiones paralelizadas de las compuertas fundamentales. Luego, se simulan dos implementaciones distintas del algoritmo de Shor para factorizar números de n bits, que utilizan $2n+3$ y $2n+2$ qubits cada una [2, 3].

Despite the discovery of many quantum algorithms with great theoretical potential, to date the main limitation of Quantum Computing in practice is the poor scalability of quantum computers. Shor's factoring algorithm [1] is one of these novel algorithms, and it allows an exponential improvement in the computational costs of factoring large integers. The complexity of the factoring problem using classical computers is the key to the strength of many of the most widely used public key cryptographic systems, and a potential scalable implementation of Shor's algorithm would make them obsolete.
In this work, a theory for the simulation of quantum algorithms on classical computers is presented and developed. From this, a numerical implementation that allows the simulation of quantum gates and circuits is carried out, introducing algebraic techniques to reduce the number of calculations by the time parallelized versions of the fundamental gates are presented. Then, two different implementations of Shor's algorithm for factoring n bits numbers are simulated, with $2n+3$ and $2n+2$ qubits each. [2, 3].

Índice general

Capítulo 1

Introducción

Aunque a la fecha su utilidad práctica sea aún poca, la Computación Cuántica es una tecnología disruptiva con potencial para cambiar radicalmente el mundo en que vivimos en los próximos años. A principio de la década de 1980, Richard Feynmann en su trabajo *Simulando Física en computadoras* [4] y Yuri Manin en *Computable y no computable* [5], propusieron por primera vez la idea y el concepto de computadora cuántica. No fue sino hasta que Peter W. Shor publicara en 1994 *Algoritmos para computación cuántica: logarítmos discretos y factorización* [6] que la disciplina alcanzo el interés general. En este trabajo, se introdujo un algoritmo cuántico de factorización de enteros en primos que ofrecía una mejora exponencial en su tiempo de cómputo frente al mejor algoritmo clásico existente [1].

Dicho interés es debido a que el algoritmo presentado por Shor, si se pudiera construir una computadora cuántica con un número de qubit *suficientemente grande*, volvería obsoletos la mayor parte de los algoritmos criptográficos de clave pública, entre ellos el sistema RSA [7] que es el más utilizado en la actualidad. La pregunta obvia que surge aquí es que sería suficientemente grande en este contexto, lo que nos lleva a explicar la principal limitación técnica de la Computación Cuántica. Por ejemplo, para romper un sistema RSA cuya clave tiene 2048 bits, un estándar utilizado ampliamente, se necesitarían del orden de 4096 qubits. Para quienes estamos acostumbrados a la enorme escalabilidad de las computadoras clásicas este número suena insignificante, pero hasta a la fecha de este trabajo la computadora cuántica más grande sólo tiene 433 qubits (IBM Osprey, 2022).

Esta, aunque sea la limitación más importante y en donde más esfuerzo se este invirtiendo, no es la única. Existe entre otros el problema del tiempo de decoherencia, que dicho de manera informal es el tiempo hasta que se pierde el control de los qubits, y el de la imposibilidad física de interacción entre todos los distintos qubits de la computadora. Muchos algoritmos requieren la interacción entre pares de qubits que en la implementación de la computadora cuántica se encuentran físicamente lejos e inconectados.

En general, para que una computadora cuántica pueda construirse en la práctica, deben cumplirse la condiciones necesarias conocidas como los *Criterios de DiVincenzo* [8]:

- Un sistema físico escalable con qubit bien definidos y caracterizados
- La capacidad de preparar el estado inicial del sistema (inicializar) a un estado simple y conocido.
- Tiempos de decoherencia suficientemente largos
- Un conjunto universal de compuertas cuánticas
- La capacidad de realizar mediciones en qubits específicos.

Según la *arquitectura* de cada procesador cuántico, es decir, el sistema físico en el que se basa y la implementación física de los qubits, las características de los mismos varía. La tecnología de qubits superconductores, que es la que se utiliza en IBM Osprey y otras computadoras con un alto número de qubits, tiene como limitación tiempos de decoherencia muy cortos. Por otra parte, otras tecnologías como los iones atrapados ofrecen tiempos de decoherencia mucho más largos, pero una escalabilidad limitada en término de número de qubits.

A la vez que se estudia como mejorar el *hardware* cuántico, también se estudia como mejorar los algoritmos cuánticos, es decir el *software*, para reducir sus requerimientos tanto en número de qubits como en tiempo de cómputo, entre otros recursos computacionales. En su trabajo, Peter W. Shor sólo introdujo su algoritmo como un caso particular del procedimiento de estimación de fase, pero una parte fundamental del mismo consistente en la implementación reversible de la exponenciación modular no fue completamente definida, si no que se la utilizó como *oráculo* o *caja negra*[1]. Varios autores [2, 3, 9-14] abordaron este problema durante los último 25 años, logrando sucesivos avances en la reducción del número de qubits y el número de compuertas cuánticas necesarias para la implementación completa del algoritmo de Shor.

Con el objetivo de desarrollar algoritmos cuánticos, resulta fundamental poder experimentar con ellos para de esta forma comprobar su funcionamiento. Para ello, es necesario una herramienta de simulación que permita definir un vector de estado a partir de muchos qubits y seguir su evolución a medida que se le aplican las compuertas cuánticas necesarias. La misma debe ser eficiente en cuanto a uso de recursos computacionales, ya que debe ser capaz de simular sistemas de un tamaño considerable.
En este trabajo se presenta la teoría para la simulación en computadoras clásicas de vectores

[1]Un oráculo o caja negra es un operador hipotético que de alguna forma no relevante para el caso implementa cierta operación.

de estado, compuertas y algoritmos cuánticos. En base a ella, se desarrolla una implementación numérica centrándose en técnicas para maximizar la eficiencia. Dichas técnicas incluyen la manipulación algebraica con el objetivo de reducir al mínimo el número de operaciones necesarias y la paralelización de las compuertas fundamentales para ser llevadas a cabo por varios procesos simultáneos. El trabajo de Nielsen & Chuang [15] ha sido ampliamente utilizado como base teórica a lo largo del trabajo.

En el capítulo 2 se da una breve introducción a los conceptos fundamentales de las Ciencias de la computación necesarias para abordar el resto del trabajo. Se presenta la lógica binaria, la teoría básica de algoritmos y los fundamentos de las Ciencias de la Computación, los modelos computacionales de Turing y de circuitos. Luego, se introducen nociones de Teoría de la Complejidad Computacional, incluyendo uso de recursos computacionales y notaciones asintóticas.

En el siguiente capítulo, 3, se desarrolla el marco teórico de la Computación Cuántica. En la sección 3.1 se presentan los conceptos matemáticos fundamentales, como vectores de estado, compuertas u operadores cuánticos y su aplicación en espacios de Hilbert complejos, y se los asocia a nociones de las ciencias de la computación como bits, compuertas y circuitos. En la sección 3.2, se desarrollan las expresiones algebraicas correspondientes a las transformaciones que sufre el vector de estado del sistema cuando se le aplican las compuertas fundamentales, y se explica como obtener operaciones más complejas usando secuencias de compuertas fundamentales.

En el capítulo 4, el más largo del trabajo, se estudia como usar estas secuencias de compuertas para llevar a cabo tareas computacionales, desde las más simples hasta llegar al algoritmo de factorización de Shor. En su introducción, 4.1, se lleva a cabo el paso de algoritmos clásicos a cuánticos y se dan los primeros ejemplos de algoritmos cuánticos, como así también se habla sobre sus potencialidades y limitaciones para resolver problemas. En la sección 4.2, se presenta la Transformada de Fourier Cuántica, QFT, versión cuántica de la transformada discreta de Fourier y una de las herramientas más poderosas de la computación cuántica. En base a la QFT se introduce el procedimiento de estimación de fase, cuyo caso particular, el algoritmo de Shor para la factorización de números enteros en sus factores primos, se presenta en la sección 4.3. Primero, se muestra como transformar el problema de factorización en el problema de hallar el orden de un elemento a en el grupo multiplicativo mod N, es decir, encontrar el período r de la función exponenciación modulo N con base a, $a^x \bmod N$, donde N es el número que se quiere factorizar y $a < N$ son coprimos entre sí. Se muestra a continuación como adaptar el procedimiento de estimación de fase al problema de hallar el orden, especificando en este último la compuerta U_a que computa la multiplicación modular y cuya aplicación sucesiva $U_a^{2^j}$ implementa la exponenciación modular. Se aborda el cuello de botella del algoritmo, la implementación completa de U_a, y finalmente se muestra como obtener r a partir del resultado de la estimación de fase.

En las siguientes secciones, se presentan dos implementaciones completas del algoritmo de Shor que incluyen las compuertas U_a, analizando su complejidad computacional. En la sección 4.4 se

presenta el algoritmo desarrollado por S. Beauregard [3] que utiliza $2n+3$ qubits para implementar completamente el algoritmo de Shor para N de n bits. En las sucesivas subsecciones se presentan las distintas operaciones necesarias, empezando por la suma de una constante a y siguiendo por la suma y la multiplicación modular, hasta llegar a U_a y la exponenciación modular. En la sección 4.4 se presenta un algoritmo que utiliza $2n + 2$ qubits, basado en el trabajo de Takahashi et al. [2] y con ayuda del trabajo de Häner et al. [9]. Aquí se presentan las operaciones diferenciales de este algoritmo con respecto al de $2n + 2$ qubits, reutilizando partes de este último.

Una vez se ha desarrollado toda la base teórica, en el capítulo 5 se aborda el problema de simular las compuertas y algoritmos fundamentales. En la sección 5.1 se explica porque se usa el lenguaje FORTRAN, con sus ventajas y desventajas, y se estudian los problemas propios de las implementaciones numéricas y la simulación de sistemas cuánticos. Luego, en la sección 5.2 se estudia el problema de la paralelización de las compuertas fundamentales y se presenta la expresiones algebraicas para dividir su cálculo en p procesos. Finalmente, en la sección 5.3 se analizan los datos que surgen de la simulación de las diferentes compuertas y algoritmos presentados a lo largo del trabajo.

En las conclusiones, 6, se discuten los resultados obtenidos a lo largo del trabajo en este proyecto y se sugieren próximas investigaciones.

Capítulo 2

Introducción a las Ciencias de la computación

El campo de la Computación Cuántica resulta un punto de intersección entre la Física y las Ciencias de la Computación, atrayendo interesados desde ambas partes. Es fundamental estar suficientemente formado tanto en Mecánica Cuántica como en Ciencias de la Computación, y específicamente dentro de esta última, en Teoría de la Complejidad Computacional, para poder adentrarse exitosamente en él.

El acercamiento del autor al campo por el lado de la física implicó la necesidad de aprender nociones de las Ciencias de la Computación, empezando por lo básico. En este capítulo se introducen los conceptos de este campo necesarios para abordar el trabajo.

En la sección 2.1 se dan nociones básicas lógica binaria. Se desarrolla el concepto de bit de información y se introducen las operaciones lógicas fundamentales que constituyen la base de todo algoritmo computacional. La sección 2.2 da las motivaciones históricas e intuitivas para el desarrollo de una teoría formal de las Ciencias de la Computación explorando su concepto fundamental: el algoritmo. Se desarrolla a lo largo de la misma el concepto de modelo de computación y se introduce la máquina de Turing como base para el desarrollo de una teoría formal. Luego, se introduce el modelo de circuitos, el cuál presenta una mayor utilidad práctica a la hora de estudiar algoritmos y será utilizado en su versión cuántica de manera intensa en este trabajo.

En la sección 2.3 se desarrolla la notación asintótica y se introducen las primeras nociones de teoría de la Complejidad Computacional, uno de los campos que fomentó el interés por la Computación Cuántica. Los conceptos desarrollados en esta sección serán utilizados luego a la hora de comparar distintos algoritmos e implementaciones.

El lector familiarizado con estos conceptos puede proceder al siguiente capítulo donde se expone y desarrolla la teoría que sustenta las posteriores simulaciones.

2.1. Lógica Binaria

La lógica binaria es uno de los fundamentos teóricos de la Computación, en tanto es la base de los llamados sistemas digitales. Como dice su nombre, sus variables solo presentan dos valores posibles, *Verdadero* o *Falso*, 0 o 1, o *encendido* o *apagado*. Este último caso es de especial utilidad por su analogía con interruptores eléctricos, ya que pueden ser fácilmente implementadas mediante circuitos eléctricos. Se deja circular corriente para uno de los estados, mientras que se corta el circuito para el otro. También, estos sistemas resultan ser fácilmente escalables, por lo que junto con la clara ventaja de la rapidez de la transmisión de información mediante impulsos eléctricos, supuso el primer impulso de las computadoras digitales.

2.1.1. Bits

La unidad mínima de información en un sistema binario es un *bit*. Este tiene dos estados, Verdadero y Falso, que podemos codificar en los números 1 y 0 respectivamente. Sin embargo, otros sistemas mas complejos son fácilmente codificables utilizando varios bits, con la única limitación de tener una precisión finita a la hora de codificar variables continuas. La escalabilidad que demostraron los circuitos eléctricos permitió superar esta dificultad mediante fuerza bruta, aumentando el número de bits y por lo tanto la precisión de la representación.

2.1.2. Operaciones Lógicas

Para procesar esta información, existen dentro de la lógica binaria las llamadas *operaciones lógicas*. Para un solo bit, existe la *negación lógica o complemento* o NOT, que cambia un bit de un valor al otro. Para dos bits de entrada, las posibilidades aumentan, por ejemplo:

- Disyunción lógica o OR: es falso sólo si ambas entradas son falsas, y verdadera en todos los demás casos.
- Disyunción lógica exclusiva o XOR: es falsa si ambas entradas son iguales.
- Conjunción lógica o AND: es verdadera sólo si ambas entradas son verdaderas, y falsa en los demás casos

A continuación se muestran las *tablas de verdad* de las operaciones lógicas.

Como la información puede codificarse en el sistema binario, los algoritmos que procesan dicha información también pueden codificarse con estas operaciones, por lo que se las llama *fundamentales*. A su vez, en un circuito electrónico estas compuertas pueden implementarse mediante dispositivos electrónicos semiconductores, como los transistores, que permiten modificar la corriente de que pasa a través de ellos según estén conectados a una segunda corriente. Luego, solo es

A	NOT A
0	1
1	0

(a) NOT

A	B	A AND B
0	0	0
0	1	0
1	0	0
1	1	1

(b) AND

A	B	A OR B
0	0	0
0	1	1
1	0	1
1	1	1

(c) OR

A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

(d) XOR

Cuadro 2.1: Operaciones Lógicas

necesario implementar mediante sistemas físicos una pequeña cantidad de compuertas para hacer posible la ejecución de una enorme cantidad de algoritmos.

2.2. Algoritmos y Modelos de Computación

El estudio de los *Algoritmos* es uno de los campos mas importantes de las Ciencias de la Computación. Todos estamos familiarizados con el concepto fundamental que un algoritmo representa, a pesar de quizás nos asociar dicho concepto a la palabra. Un algoritmo no es mas que una receta, precisa y concreta, para realizar una cierta tarea. Un ejemplo ilustrativo podría ser el algoritmo elemental para sumar dos números que todos aprendemos en la escuela primaria. Pero, ¿cuál es la definición matemática precisa de un algoritmo? Por siglos, esta pregunta ha sido dejada de lado en favor del concepto intuitivo de receta, aunque procedimiento perfectamente clasificables (y usados hasta nuestros días) ya se estuvieran desarrollando. El algoritmo de Euclídes para hallar máximo divisor común, por ejemplo, cumple todos los supuestos posteriormente definidos, a pesar de haber sido descubierto hace mas de dos mil años. No fue hasta principios del siglo XX que se empezó a avanzar en el camino de definir rigurosamente que es un algoritmo. David Hilbert, un reputado matemático alemán, planteo una desafiante pregunta: ¿Existe, al menos en teoría, un algoritmo capaz de resolver *cualquier* problema matemático? 1

En respuesta a esto es que grandes pioneros como Alan Turing, Alonzo Church y demás, desarrollaron los primeros conceptos de la teoría moderna de los algoritmos. La pregunta de Hilbert, quien la bautizó como *Entscheidungsproblem*, alemán para 'problema de decisión´, resulto para su sorpresa tener una respuesta negativa. Para probar esto, Turing y Church tuvieron que enfrentarse al problema de definir rigurosamente a los algoritmos como un concepto matemático que capturase el concepto intuitivo que ya existía. En su camino, sentaron las bases de la teoría moderna de los algoritmos y por consiguiente, de la teoría de las Ciencias de la Computación.

Thesis 1 (Problema de Decisión de Hilbert)**.** *Existe un algoritmo capaz de resolver cualquier problema matemático*

En este capítulos, se exploraran dos modelos teóricos de Computación distintos: *La Máquina de Turing* y el Modelo de Circuitos. Ambos modelos son capaces de simularse el uno al otro, y ofrecen diferentes perspectivas únicas que los hacen por igual utiles a la hora de enfrentar problemas computacionales.

2.2.1. Máquinas de Turing y el Problema de Decisión

Las máquinas de Turing son construcciones teóricas tan poderosas como simples, ya que constan de solo cuatro elementos principales: 1) un centro de control de estado finito, 2) una cinta, 3) un cabezal de lecto-escritura y 4) un *programa*. En detalle:

1. El centro de control de estado finito consiste en un conjunto finito de *estados internos* $A = q_1, \ldots, q_m$. Básicamente, desarrolla las funciones de un procesador, coordinando la operativa de la máquina de Turing, proporcionando también cierto almacenamiento temporal. Existen dos estados especiales que determinan el comienzo y el final de la ejecución de la máquina, siendo estos q_s y q_h respectivamente.

2. La cinta es un objeto unidimensional semi infinito que cumple la función de almacenamiento. Consiste en una sucesión ordenada de espacios, cada cuál contiene un cierto símbolo de un conjunto finito de estos, llamado *alfabeto*Λ. Por simplicidad, consideraremos cuatro simbolos, 0, 1, b y $\triangleright$. Este último indica el inicio de la cinta, mientras que b indica un espacio vacío.

3. El cabezal se encarga de identificar, leer y reescribir los simbolos contenidos en los espacios de la cinta. Inicialmente, el cabezal apunta al primer espacio, aquel que contiene el simbolo $\triangleright$.

4. Finalmente, el programa es una lista ordenada de *lineas* de la forma $\langle q, x, q', x', s\rangle$, donde $q, q' \in A$ y $x, x' \in \Lambda$.

La ejecución del programa se da de la siguiente forma: supongamos que el estado interno de la máquina es q y el cabezal se encuentra en un espacio cuyo simbolo es x. Entonces, se busca entre las lineas del programa una que contenga $\langle q, x, \ldots\rangle$. En caso de no encontrar ninguna, se cambia el estado interno a q_h y el programa se detien. Si se encuentra una linea así, entonces esta se *ejecuta* cambiando el estado interno q por q' y el simbolo x por x'. Luego, dependiendo si $s = -1, 1o1$, se desplaza el cabezal al espacio previo, permanece en el mismo, o se mueve al siguiente, excepto cuando el cabezal se encuentra en el comienzo de la cinta donde no puede moverse más hacia a la izquierda. Veamos con un ejemplo simple como una máquina de Turing puede ser usada para computar una función. Al comenzar la ejecución del programa, la cinta contiene el número binario x, seguido de espacios en blanco. La máquina consta de tres estados internos q_1, q_2 y q_3, además de q_s y q_h. El programa es el siguiente:

1. $\langle q_s, \triangleright, q_1, \triangleright, 1\rangle$
2. $\langle q_1, 0, q_1, b, 1\rangle$
3. $\langle q_1, 1, q_1, b, 1\rangle$
4. $\langle q_1, b, q_2, b, -1\rangle$
5. $\langle q_2, b, q_2, b, -1\rangle$
6. $\langle q_2, \triangleright, q_3, \triangleright, 1\rangle$
7. $\langle q_3, b, q_h, 1, 0\rangle$

Al inicio, al estar la máquina en el estado q_s y el cabezal apuntando al primer espacio de la cinta, el cuál esta ocupado por $\triangleright$, se ejecuta la primera linea, que cambia el estado a q_1 y mueve el cabezal hacia el siguiente espacio. En este espacio, el símbolo contenido es la mayor cifra binaria de x, que por definición es un 1, por lo que al estar la máquina en estado q_1 se ejecuta la tercera linea, borrando el simbolo 1 (es decir, cambiandolo por b)y desplazando el cabezal al siguiente espacio. Mientras la cinta contenga 0 o 1 en sus siguientes espacios, se ejecutaran las lineas 2 y 3 respectivamente, borrando cada simbolo a medida que el cabezal se desplaza. Cuando el cabezal supere la ultima cifra binaria de x, se encontrara con espacios vacios, por lo que ejecutara la linea 4, que retrocederá un espacio a la vez que cambia el estado del sistema a q_2. Al retroceder, se encontrará con un espacio recién borrado, por lo que ejecutará la linea 5, retrociendo un espacio sin cambiar ni el estado ni el simbolo. Esto se repite hasta llegar al inicio de la cinta, cuyo simbolo $\triangleright$ aún permanece. Se ejecuta entonces la linea 6, cambiando el estado a q_3 y moviendo el cabezal al sugiente espacio. Finalmente, al encontrarse otra vez con un espacio vacio, se ejecuta la linea 7, escribiendo 1 en el espacio y cambiando el estado a q_h, lo cuál implica que el programa ha terminado. Como se puede ver, ahora en la cinta se encuentra almacenado un 1 seguido de espacios vacíos, que se corresponde con $x = 1$. Por lo tanto, el programa computa la función $f(x) = 1$, para cualquier $x \in \mathbb{R}$.

Tesis de Church-Turing

La siguiente pregunta surge de manera directa: ¿Que clase de funciones es posible computar con una máquina de Turing? Como se ha anticipado, el modelo de computación de las Máquinas de Turing permite el computo de una enorme cantidad de tareas, incluyendo todas las operaciones aritméticas básicas, algoritmos de búsqueda y ordenamiento, entre otras. De hecho, es tal el poder de éstas que resultan ser capaces de simular cualquier operación de una computadora moderna. Y es aquí donde retornamos a la cuestión inicial de definir formalmente que es un algoritmo. Church y Turing formularon independientemente la siguiente tesis:

Thesis 2 (Tesis de Church-Turing)**.** *La clase de funciones computables por una máquina de Turing se corresponde exactamente con la clase de funciones computables por un algoritmo.*

Básicamente, esta tesis establece la equivalencia entre un concepto matemático riguroso como es la máquina de Turing, y el concepto intuitivo de algoritmo. Vale la pena notar por qué se dice que es una tesis y no un teorema. Esto es debido a que los conceptos intuitivos de procedimiento efectivo y algoritmo no son conceptos de ninguna teoría matemática ni existe una definición aceptada, por lo que la tesis no puede ser probada rigurosamente. Aunque se presume como cierta, solo bastaría un contraejemplo para demostrarla errónea. Esto último se antoja improbable, y la validez de la tesis de Church-Turing cuenta con un amplio consenso en las ciencias de la computación. Al establecer la equivalencia entre algoritmos y máquinas de Turing, la Tesis de Crunch-Turing permite el estudio de los primeros de una manera matemática y rigurosa, y de esta manera poder construir un marco teórico completo.

Número de Turing y Máquina de Turing Universal

Se puede demostrar a que cualquier máquina de Turing puede asignarse un número natural de manera univoca. A este número se le denomina *Número de Turing*. En base a este concepto, puede definirse la llamada *Máquina de Turing Universal*. Sea M una máquina de Turing que calcula una operación $M(x)$ y sea T_M su número de Turing. Supongamos entonces que existe una máquina tal que a una cinta de entrada con la representación binaria de T_M seguida después de un espacio por el conjunto de simbolos x que eran originalmente la entrada de la máquina M, compute la salida de la operación $M(x)$. Entonces esta máquina, la Máquina de Turing universal, es capaz de simular cualquier otra máquina M. Puede demostrarse que es posible construir de manera rigurosa la Máquina de Turing Universal, capaz de computar cualquier algoritmo.

Problema de Decisión y Problema de la parada

Es utilizando la máquina Universal de Turing que puede demostrarse la falsedad del problema de decisión de Hilbert. Para esto, Turing utilizo el *Problema de la Parada*: Dada la máquina de Turing número x, ¿se detendrá alguna vez ante un numero de entrada y?. Definamos la función de parada:

$$h(x) = \begin{cases} 0 & \text{si la máquina número } x \text{ no se detiene ante una entrada } y \\ 1 & \text{si la máquina número } x \text{ se detiene ante una entrada } y \end{cases} \tag{2.1}$$

Supongamos existe un algoritmo para resolver el problema de la parada, $HALT(x)$, lo cuál sucede si y solo si existe uno para evaluar $h(x)$, y consideremos el siguiente pseudocódigo:

Algorithm 1 Problema de la parada

```
1: function TURING(x)
2:     y = HALT(x)
3:     loop
4:         if y = 0 then
5:             break
6:         end if
7:     end loop
8: end function
```

Por la tesis de Church-Turing, sabemos la función $TURING$ tiene un número de Turing t. Luego, $h(t) = 1 \iff TURING$ se detiene para t. Sin embargo, podemos observar que $TURING$ se detiene para $t \iff h(t) = 0$. Por lo tanto, $h(t) = 0 \iff h(t) = 1$. Obtenemos entonces que nuestra suposición inicial de la existencia de un algoritmo para evaluar $h(x)$ debe ser falsa, lo que implica que no existe ningún algoritmo que nos permita resolver el problema de la Parada. Con este contraejemplo, que por cierto no es el único hallado hasta ahora, demostramos que no existe algoritmo capaz de resolver todos los problemas matemáticos.

2.2.2. Modelo de Circuitos

En esta sección exploraremos el modelo de circuitos, que proporciona un punto de vista mas realista y práctico a la hora de aplicarlo a computadoras reales, ya que a diferencia de las máquinas de Turing, no presupone un tamaño ilimitado. Un *circuito* esta compuesto de *cables* y *compuertas*, que transportan información y realizan tareas de cómputo simples, respectivamente. En el circuito de la operación NOT, es decir, la negación lógica de un bit, el bit a ingresa por la izquierda, es transportado por el cable hasta la compuerta NOT donde su valor cambia de 0 a 1, o de 1 a 0.
Generalizando, un circuito puede incorporar muchos bits de entrada y de salida, que no necesariamente tienen que ser en la misma cantidad, y muchas compuertas lógicas. Una compuerta lógica implementa una función $F : \{0,1\}^n \to \{0,1\}^m$, de n bits de entrada en m bits de salida. La compuerta NOT de arriba, implementa la función $F(a) = 1 \oplus a$ con a el bit de entrada y $\oplus$ la suma modulo 2.
Otras compuertas lógicas elementales son AND, OR y XOR, las cuales representan las operaciones lógicas vistas en la sección 2.1 Y, O y O exclusivo respectivamente. En la figura 2.1, se muestran los circuitos correspondientes. Hay dos compuertas importantes en el contexto de los circuitos, pero que no implementan operaciones lógicas, estas son $FANOUT$ que crea copias un bit, básicamente dividiendo su cable, y $CROSSOVER$ o $SWAP$, que intercambia el valor de dos bits.

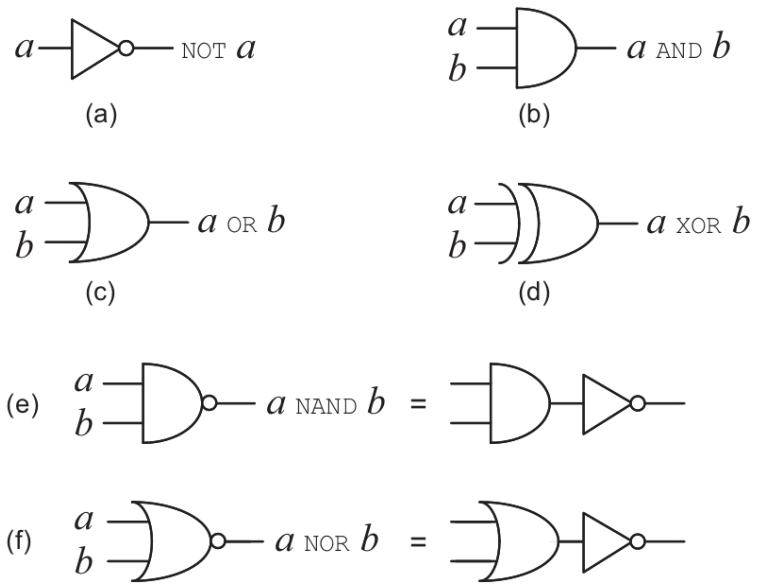


Figura 2.1: Circuitos de las operaciones lógicas fundamentales: a) NOT, b) AND, c) OR y d) XOR.
También se incluyen las operaciones e) $NAND$ y f) NOR que son AND y OR respectivamente, pero seguidas de un NOT

Usando estas compuertas elementales, se pueden construir recursivamente una enorme cantidad de tareas computacionales. Por ejemplo, veamos un circuito básico para sumar enteros de n bits. Primero, el circuito *half adder* (HA), de la figura 2.2, implementa la suma binaria de dos bits de entrada, x e y, obteniéndose a su salida el bit de acarreo c que vale 1 si y solo si tanto x como y valen 1, y $s = x \oplus y$.

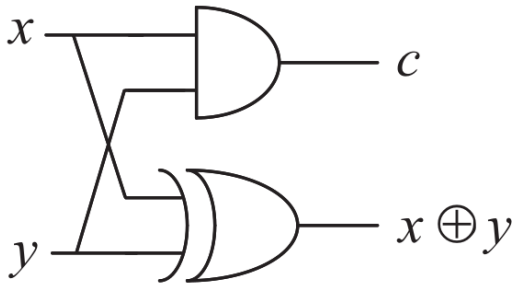


Figura 2.2: Circuito del sumador de un bit *half adder*.

Combinando dos *half adder* se obtiene un *full adder* (FA), figura 2.3, que tiene en cuenta un bit de acarreo procedente de un cálculo anterior. Este circuito presenta tres entradas, x, y y c, y

dos de salida, c' y s e implementa la siguiente operación:

$$\begin{aligned} s &= x \oplus y \oplus c\oplus \\ c' &= (x \cdot y) + c \cdot (x \oplus y) \end{aligned} \tag{2.2}$$

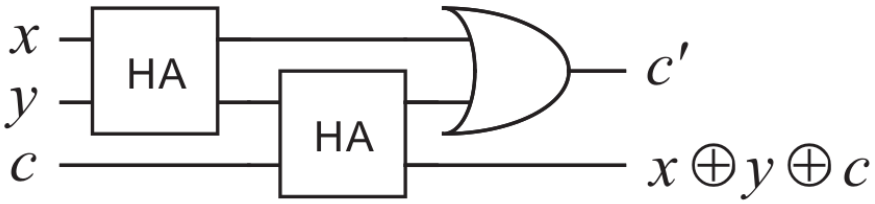


Figura 2.3: Circuito del sumador de un bit *full adder*, que tiene en cuenta un bit de acarreo de entrada.

Luego, es directo construir la suma binaria de n bits encadenando half adders y full adders, como se muestra en la figura 2.4) para $n = 3$.

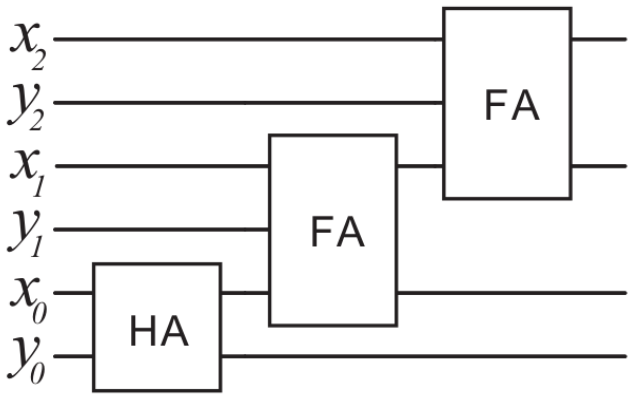


Figura 2.4: Circuito de un sumador de tres bits, usando dos FA y un HA.

Como los circuitos son un modelo de computación, estos deben ser capaces de simular cualquier función $F : \{0,1\}^n \to \{0,1\}^m$. Empecemos por probar el caso en que $m = 1$, es decir n bits de entrada y uno de salida. Estas funciones se denominan *Funciones Booleanas* y su correspondiente circuito *Circuitos Booleanos*. La prueba es por inducción en n. Para $n = 1$ existe solo 4 posibles funciones, que son fáciles de demostrar ser computables por circuitos, con la ayuda de un bit extra en un estado conocido:

$$F_0(x) = x \quad F_1(x) = -x \quad F_2(x) = 1 \quad F_3(x) = 0 \tag{2.3}$$

Para el paso inductivo, supongamos que cualquier función de n bits puede ser computada por un circuito, pasemos la caso $n+1$. Definamos las siguientes funciones: $f_0(x_1, \ldots, x_n) \equiv f(0, x_1, \ldots, x_n)$ y $f_1(x_1, \ldots, x_n) \equiv f(1, x_1, \ldots, x_n)$. Por hipótesis, tanto f_0 como f_1 son computables con circuitos, por lo que usando sus circuitos es directo construir el de la función f de $n+1$ bits. Para ello, primero se computan tanto f_0 como f_1 con los últimos n bits, y luego, según x_0 sea 0 o 1, se elige el valor correspondiente para f. En el circuito 2.5, se muestra el circuito donde se utilizan cajas negras que computan las funciones f_0 y f_1 Esto completa la demostración inductiva.

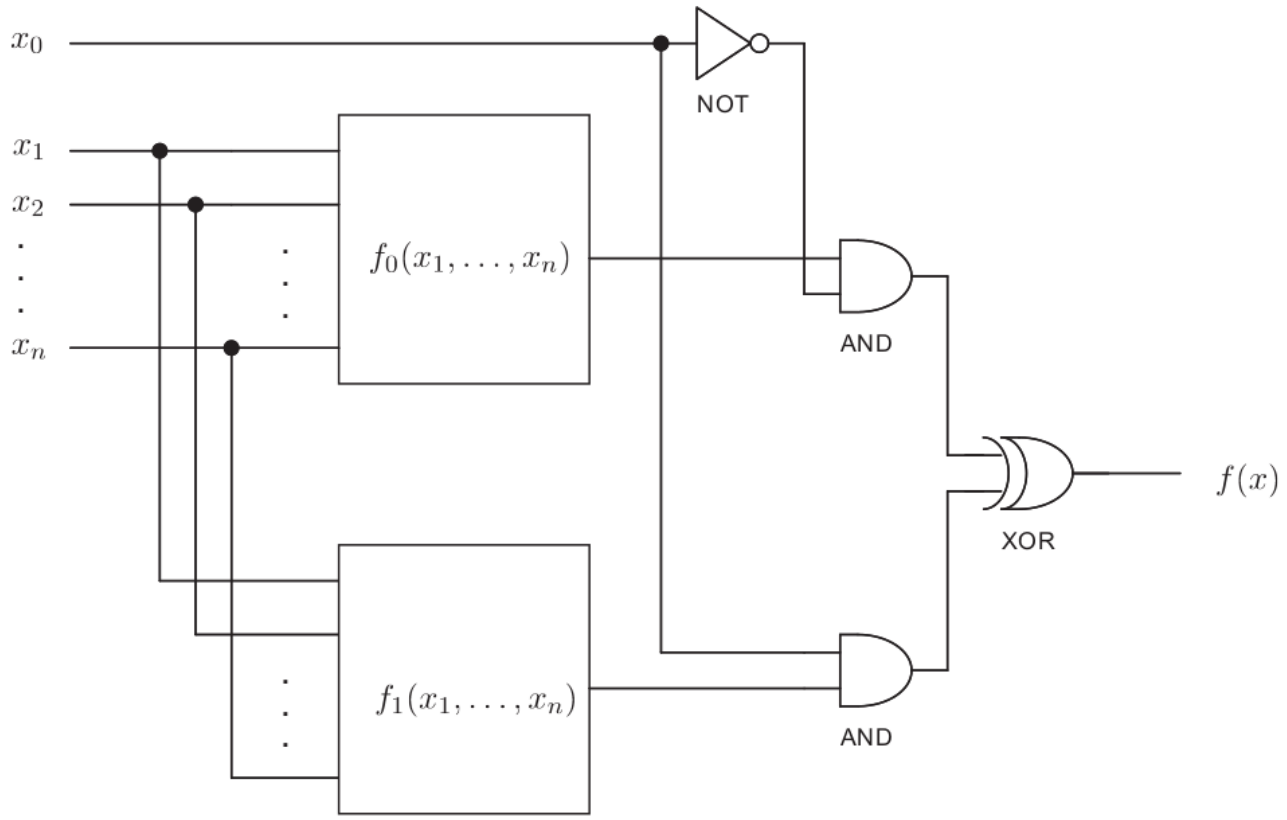


Figura 2.5: Circuito que computa la función f de $n+1$ bits asumiendo que existen circuitos para calcular las funciones de n bits f_0 y f_1.

Para el caso general con $m \neq 1$, simplemente se repite en el paso inductivo m veces el circuito, una por cada cifra binaria de la salida. Hay que tener en cuenta que para $n = 1$ ya no son 4 funciones posibles, pero se puede demostrar que todas son plausibles de ser computadas mediante circuitos.

En esta demostración se utilizaron cinco elementos fundamentales, cables, que marcan el flujo de información, bits de trabajo o *ancilla* (*ancillae*), la operación $FANOUT$, la operación $SWAP$ y las compuertas lógicas AND, OR y NOT. Con estos elementos, puede construirse el circuito

para computar cualquier función binaria. Esto es útil ya que un procesador físico podría ser capaz de computar cualquier función solo implementando estos cinco elementos.

2.3. El análisis de los Problemas Computacionales

¿Qué es un problema *computacional*? A priori, no parece un pregunta de mayor relevancia, pero como en el caso de los algoritmos, la definición precisa de problema computacional tiene sus matices. Con el objetivo de desarrollar un marco teórico mas general, es conveniente restringirnos primero a una clase particular de problemas llamados *problemas de decisión* en analogía con la pregunta de Hilbert. Luego, dichos principios pueden extrapolarse a problemas mas generales, incluyendo en estos los propios de la computación cuántica. Aunque en teoría un algoritmo sea capaz de hacer operaciones mas o menos interesantes, en la práctica su utilidad esta limitada a la plausibilidad del mismo de ser implementado en la realidad con procesadores físicos. Dicha implementación de un algoritmo consume recursos, como tiempo de cómputo, espacio de almacenamiento y energía. Esta claro que es deseable, en lo que respecta a utilidad práctica, reducir el consumo de dichos recursos. Para este fin, se necesita un marco teórico para la cuantificación de los recursos utilizados por los algoritmos que sean inherentes a los mismos, es decir, no dependan de una implementación física particular.

2.3.1. Cuantificación de los Recursos Computacionales: Notación Asintótica

Diferentes modelos de computación conllevan diferentes requerimientos en cuanto a recursos. Esta claro que las máquinas de Turing originales que poseen un cinta infinita, requerirían un espacio de almacenamiento infinito, lo cuál no es posible. Incluso en el cambio de un modelo de una sola cinta a otro de dos cintas, los requerimientos de recursos varían. Como lo que en general se quiere estudiar son los problemas computacionales en si, no los modelos, se necesita una manera de cuantificar los recursos que sea independiente también de los modelos de computación particulares. O sea, lo *esencial* al comportamiento del problema en sí.

Una herramienta útil es el uso de la *notación asintótica*, que básicamente consiste en acotar o dar aproximaciones de la cantidad de recursos que son inherentes a un problema computacional, sin preocuparse por obtener números exactos. A continuación se describirá esta notación, aplicándose la misma a un problema particular para ejemplificar su uso.

La notación O ('big O') es usada para dar cotas superiores al comportamiento de una función. Se dice que una función $f(n)$ 'esta en la clase de funciones $\mathcal{O}(g(n))$' o simplemente 'es $\mathcal{O}(g(n))$ si vale que $f(n) \leq cg(n)\ \forall n > n_0$ para ciertas constantes c y n_0. Es decir, $g(n)$ es una cota

superior de $f(n)$ para n suficientemente grande, a menos de una constante. La notación big O es particularmente útil para estudiar el peor caso de comportamiento para un algoritmo *específico*.

Cuando estudiamos un clase de algoritmos, por ejemplo, todos los algoritmos posibles para multiplicar dos números, resulta interesante dar un cota inferior a los recursos. Para esto, se utiliza la notación Ω ('big Omega'). Se dice que '$f(n)$ es $\Omega(g(n))$' si existen constantes c y n_0 tales que para n mayor que n_0, $f(n) \geq cg(n)$. Es decir, $g(n)$ es una cota inferior de $f(n)$ para n suficientemente grande.

Por último, la notación Θ ('big Theta') implica el mismo comportamiento asintótico. O sea, '$f(n)$ es $\Theta(g(n))$' significa que $f(n))$ es tanto $\mathcal{O}(g(n))$ como $\Omega(g(n))$

Veamos ejemplos. La función $f(n) = 2n$ cumple $2n \leq 2n^2$ para todo n, por lo que decimos que f es de clase $\mathcal{O}(n^2)$. También, como $n^3 > 2^n$ para $n \geq 3$, podemos decir que 2^n es $\mathcal{O}(n^3)$. Por último, vemos que

$$\lim_{n\to\infty} \frac{5n^2 + 3n + \sqrt{n} + 2log(n^2)}{5n^2} = \lim_{n\to\infty} \frac{5n^2}{5n^2} = 1 \tag{2.4}$$

Ahora, aplicaremos la notación asintótica para cuantificar recursos computacionales, usando el ejemplo del problema de ordenar un lista de nombres con n elementos en orden alfabético. Muchos algoritmos para realizar esta tarea se basan en la operación *compara e intercambia*, es decir, tomar los elementos de a pares, compararlos e intercambiarlos si están en el orden alfabético incorrecto. Usando esto, ¿cuántas operaciones son necesarias para asegurar que la lista este ordenada? A continuación se muestra una algoritmo simple para resolver el problema de ordenamiento, donde `compareandswap(j,k)` compara e intercambia de ser necesarios los elementos j y k:

Algorithm 2 Compara e Intercambia

```
1: for j = 1, ..., n − 1 do
2:     for k = j + 1, ..., n do
3:         compareandswap(j,k)
4:     end for
5: end for
```

Como podemos ver, el total de operaciones `compareandswap(j,k)` es $(n-1) + (n-2) + \cdots + 1 = n(n-1)/2$. Por lo tanto, podemos decir que el numero de operaciones usadas por el algoritmo es $\Theta(n^2)$. Existen algoritmos mas complejos para ordenar alfabéticamente que reqieren solo $\mathcal{O}(nlog(n))$ operaciones `compareandswap(j,k)`. Es más, se puede demostrar también que todo algoritmo que use `compareandswap(j,k)` requiere $\Omega(nlog(n))$ de estas, por que en general, podemos decir que el problema de ordenamiento alfabético es de clase $\Theta(nlog(n))$ en el numero de operaciones `compareandswap(j,k)`.

2.3.2. Introducción a la Teoría de la Complejidad Computacional

Como ya hemos dicho, saber que recursos son necesarios para cierta tarea computacional es de la información más valiosa a nivel práctico, ya que de ello depende si vale o no la pena llevarla a cabo. Para problemas como la adición y la multiplicación se considera que es eficiente resolverlos porque existen algoritmos *rápidos* para llevar a cabo dichas tareas, que consumen *poco* espacio. Por otro lado, para otros problemas se que son efectivamente irresolubles, no porque no existan algoritmos para hacerlo, si no porque en la práctica estos son inaplicables debido a la cantidad de recursos necesarios.

En este contexto, la *Complejidad Computacional* es el estudio de los recursos de tiempo y espacio necesarios para resolver problemas computacionales. Su tarea es obtener cotas inferiores, o sea, $\mathcal{O}(g(n))$ a la cantidad de recursos requeridos por *el mejor algoritmo posible* para resolver un problema computacional, a pesar de este poder no ser conocido explícitamente.

Una dificultad fundamental al momento de formular la Teoría de la Complejidad Computacional, es que como dijimos a principio de la sección anterior, distintos modelos computacionales requieren distinta cantidad de recursos para resolver el mismo problema. Para salvar esto, encaramos el problema desde un punto de vista más amplio y general. Si el problema requiere una entrada de cierto tamaño, cuantificable mediante un número n, el objetivo principal es distinguir si la cantidad de recursos requerida por dicho problema pueden ser acotados por una función *polinómica* en n o no. En caso negativo, decimos que los recursos crecen o *escalan* de forma *exponencial*, abusando de este término para incluir cualquier función que crece más rápido que cualquier polinomio, como por ejemplo $n^{log(n)}$. Los problemas que pertenecen a la primera clase se consideran *fáciles*, *manejables* o *feasibles*, mientras que a los segundos se les llama *difíciles*, *inmanejables* o *no feasibles* si el mejor algoritmo posible para resolverlos requiere un cantidad exponencial de recursos. Aquí es donde entra el problema de la factorización de un entero en sus factores primos, pues se cree, aunque nunca fue probado, que este problema es inmanejable. Es decir, no existe ningún algoritmo capaz de factorizar un entero de n bits arbitrario usando $\mathcal{O}(p(n))$ operaciones, para $p(n)$ algún polinomio.

Una razón fundamental para basar el estudio de la complejidad complejidad en la clasificación polinomial o exponencial se debe a la llamada *Tesis de Church-Turing fuerte*:

Thesis 3 (Tesis de Church-Turing fuerte)**.** *Todo modelo de computación puede ser simulado en una Máquina de Turing Probabilística con a lo más un aumento polinómico en el número de operaciones requeridas.*

Esta tesis permite centrarnos solo en un modelo computacional, el de las máquinas de Turing probabilísticas, lo cuál resulta de gran ayuda a la teoría de la complejidad computacional. Luego, si un problema no tiene solución polinómica en este modelo, no la tendrá en ningún otro. Sin

embargo, la computación cuántica ha demostrado recientemente la existencia de soluciones eficientes a problemas que se creían inmanejables para cualquier modelo de computación incluidas las máquinas de Turing probabilísticas, como el ya mencionado algoritmo de factorización de Shor [1], el cuál representó uno de sus mayores impulsos en su momento. Es útil de todas formas conocer y entender la tesis de Turing fuerte y su rol en el desarrollo de la teoría de la complejidad computacional.

Además de este interés teórico en clasificar los problemas computacionales en polinomiales o exponenciales, en el estudio de los algoritmos y en general las Ciencias de la Computación, existe el obvio interés en la mejora de los algoritmos aunque dicha mejora no implique pasar de recursos exponenciales a polinomiales. Por ejemplo, a pesar de ambos algoritmos ser polinomiales, uno que requiera $\Theta(n^2)$ tiempo es el doble de rápido que uno que requiera $\Theta(2n^2)$, y ni hablar de otro que requiera $\Theta(n^3)$. Pero incluso así no todo está dicho, porque en la práctica se trabaja con entradas de tamaño finito, y a pesar de que en límite cierto algoritmo sea mejor que otro, quizás no sea el caso para n finitos, por ejemplo $2n^2 > n^3$ si $n = 1$.

Capítulo 3

Computación Cuántica

En base a los conceptos de la computación clásica desarrollados en el capítulo anterior, nos abocaremos ahora a transitar la intersección que representa la Computación Cuántica desde el otro camino, la Mecánica Cuántica. Si bien no se presentará este campo desde sus fundamentos teóricos, es decir, los Postulados de la Mecánica Cuántica, se abordarán los elementos teóricos necesarios para este trabajo de una forma práctica a medida que los necesiten los conceptos propios de la Computación Cuántica. De todas formas, y teniendo en cuenta la formación del autor como físico, se asumirá una mínima experiencia en el estudio de Mecánica Cuántica. El lector interesado puede consultar (referencias).

El capítulo cuenta con dos secciones. La sección 3.1 comienza desarrollando la unidad de información cuántica, el qubit, a partir del bit clásico. Luego se presenta el marco teórico abstracto que permite el tratamiento formal de los qubits, los espacios de Hilbert y sus vectores de estado, como así también se introducen los operadores o compuertas cuánticas y las herramientas matemáticas necesarias para tratar con sistemas de muchos qubits. Por último, se expone el marco teórico y práctico de las mediciones y se da una introducción a los Circuitos Cuánticos. La segunda sección 3.2 trata íntegramente sobre los operadores cuánticos. A partir de sus formas matriciales se deducen sus expresiones como producto externo, forma que permitirá una posterior implementación numérica eficiente.

3.1. Introducción a la Computación Cuántica

Antes de adentrarse en el mundo de los algoritmos cuánticos y sus potencialidades, es necesario el conocimientos de los conceptos en los que se fundamentan. Así como en computación clásica la información codificada en bits es transformada mediante compuertas siguiendo las leyes del Álgebra Booleana, en la computación cuántica la información codificada en *qubits* se transforma mediante compuertas cuánticas, siendo estos representados por vectores de estado y operadores

unitarios, respectivamente, en espacios de Hilbert complejos. Luego, y al igual que en computación clásica, secuencias de estas compuertas se utilizan para construir algoritmos que como se verá en el capítulo 4, presentan características y utilidades sumamente interesantes.

3.1.1. Computación clásica

Como ya hemos visto en el capitulo anterior, en computación clásica la unidad fundamental de información es el bit. Un bit es un sistema físico que tiene dos estados posibles que usualmente se denotan como 1 y 0 respectivamente. La información a procesar es codificada en sistema binario y guardada en estos bits (enormes cantidades de ellos). Luego, siguiendo la reglas de la lógica binaria, estos bits son transformados según se requiera mediante las operaciones lógicas. Dichas compuertas actúan sobre los bits cambiando su valor lógico (0 o 1) según el estado inicial en el cuál estos entraron en dichas compuertas. Las compuertas *fundamentales* son aquellas a partir de las cuáles pueden construirse todas las demás. Estas son la compuerta NOT, negación o complemento, la cuál actúa en un solo bit, la compuerta AND o conjunción lógica y la compuerta OR o disyunción inclusiva. Otra compuerta importante mas no fundamental, es la compuerta XOR o disyunción exclusiva 2.1.

3.1.2. Qubits

En computación cuántica, y en analogía a la computación clásica, la unidad básica de información es el llamado *qubit*. Un qubit es un sistema físico de dos niveles de energía discretos, siendo 0 el nivel fundamental y 1 el excitado. Aquí es donde entra en juego las particularidades de la mecánica cuántica: estos qubits pueden no solo estar en el estado 0 o 1, sino en cualquier *superposición* de ellos.

Vectores de estado

El estado de un qubit puede representarse mediante un vector complejo en un espacio de Hilbert de dimensión 2, $\mathbb{C}^2$. Utilizando la notación *bra-ket*, un estado arbitrario de un qubit es $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ donde $\alpha, \beta \in \mathbb{C}$ y $|0\rangle$ y $|1\rangle$ representan los estados fundamental y excitado respectivamente, ambos de modulo unitario. Estos vectores de estado $|0\rangle$ y $|1\rangle$ constituyen una *base* para un sistema de dos niveles, es decir, un estado arbitrario de un qubit puede representarse como combinación lineal de ellos. Cualquier conjunto mínimo de estados a partir de los cuáles pueda expresarse un estado arbitrario es una base. El conjunto $\{|0\rangle, |1\rangle\}$ es llamado *base computacional*.

Siguiendo la interpretación de Copenhague, los coeficientes α y β representan la probabilidad de encontrar el sistema en los estados fundamental y excitado. Es decir, $P(|\psi\rangle = |0\rangle) = |\alpha|^2 = \alpha\alpha^*$, donde α^* denota el complejo conjugado de α. Como la suma de las probabilidades de todos los

resultados posibles tiene que ser 1, es decir, $1 = |\alpha|^2 + |\beta|^2$, se sigue que $P(|\psi\rangle = |1\rangle) = |\beta|^2 = \beta\beta^* = 1 - \alpha\alpha^*$.

Otra restricción en cuanto a los coeficientes α y β viene del hecho de la imposibilidad de distinguir dos estados con sólo una diferencia de fase global, es decir, el caso en que $|\chi\rangle = e^{i\phi}|\psi\rangle$ donde $\phi \in \mathbb{R}$. Como las probabilidades de medir cada elemento de la base dependen sólo del modulo de los coeficientes, ambos estados $|\chi\rangle$ y $|\psi\rangle$ son indistinguibles y por lo tanto, son considerados como el *mismo* estado. En base a ambas restricciones, es posible representar inequívocamente todos los estados posibles de un qubit mediante solo dos coeficientes reales.

$$|\psi\rangle = cos\Big(\frac{\theta}{2}\Big)|0\rangle + e^{i\varphi} sin\Big(\frac{\theta}{2}\Big)|1\rangle \tag{3.1}$$

Los coeficientes $\varphi, \theta \in \mathbb{R}$ definen inequívocamente un punto en una esfera tridimensional de radio 1, llamada *Esfera de Bloch*.

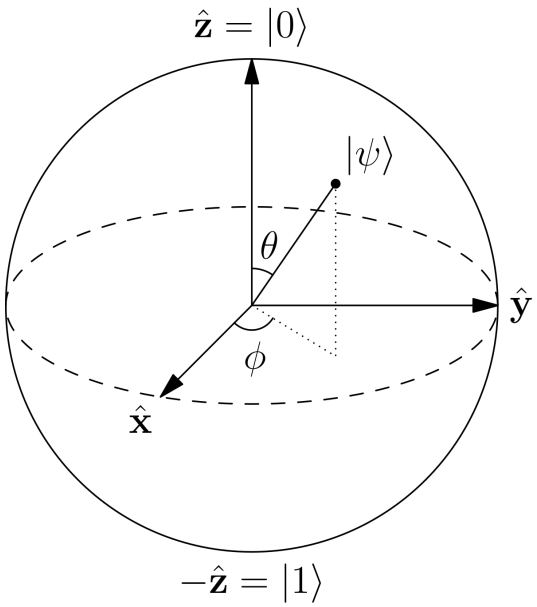


Figura 3.1: Esfera de Bloch

Como cada punto en la esfera de Bloch representa un estado distinto, uno se ve tentado a pensar que se puede codificar una cantidad infinita de información. Sin embargo, la cantidad de información extraíble de un qubit es mucho menor. Como lo único que podemos observar de un qubit es del resultado de la medición, cuyas probabilidades están dadas por los coeficientes α y β, necesitaríamos medir una cantidad infinita de qubits idénticos para poder determinar exactamente dichos coeficientes, ya que luego de la medición cada qubit colapsa al estado medido, perdiendo la información sobre α y β.

Sin embargo, aunque no la podamos medir, la información correspondiente a los coeficientes

esta allí y evoluciona junto con el sistema mientras el mismo no sea medido. Esta enorme cantidad de información *oculta* es la base del poder de cómputo de la computación cuántica.

Sistemas de varios qubits

Supongamos que se tienen dos bits. Los diferentes valores posibles son 00, 01, 10 y 11. En el caso cuántico, la base computacional de un sistema de dos qubits es $\{|00\rangle, |01\rangle, |10\rangle, |11\rangle\} = \{|\mathbf{0}\rangle, |\mathbf{1}\rangle, |\mathbf{2}\rangle, |\mathbf{3}\rangle\}$, y por lo tanto se puede representar un estado arbitrario como

$$|\psi\rangle = \sum_{x,y=0}^{1} \alpha_{xy} |xy\rangle = \sum_{x=0}^{3} \alpha_x |x\rangle \tag{3.2}$$

donde los coeficientes $\alpha_x \in \mathbb{C}$ cumplen la condición $1 = \sum_{x=0}^{3} |\alpha_x|^2$.

Ahora es posible medir no solamente todo el sistema, sino sólo uno de los qubits. Por ejemplo, suponiendo que se mide el primer qubit, observemos que los vectores de la base con el primer qubit en el estado $|0\rangle$ son

$$|00\rangle = |\mathbf{0}\rangle \qquad |01\rangle = |\mathbf{1}\rangle \tag{3.3}$$

Por lo tanto, la probabilidad de medir 0 al medir el primer qubit es $|\alpha_0|^2 + |\alpha_1|^2$. Luego de la medición, el sistema colapsa al estado

$$|\psi\rangle = \frac{\alpha_{00} |00\rangle + \alpha_{01} |01\rangle}{\sqrt{|\alpha_{00}|^2 + |\alpha_{01}|^2}} = \frac{\alpha_{\mathbf{0}} |\mathbf{0}\rangle + \alpha_{\mathbf{1}} |\mathbf{1}\rangle}{\sqrt{|\alpha_{\mathbf{0}}|^2 + |\alpha_{\mathbf{1}}|^2}} \tag{3.4}$$

donde es renormalizado al dividirlo por la raíz cuadrada de la probabilidad de medir el resultado medido. Aunque el sistema se encuentre ahora en una superposición de estados, el mismo esta *definido* a estar en el estado fundamental del primer qubit.

Una propiedad interesante de los estados de varios qubits es la posibilidad de realizar *mediciones indirectas*. En el caso de dos qubits, partiendo de un estado inicial llamado *Estado de Bell*

$$|\Phi^+\rangle = \frac{|00\rangle + |11\rangle}{\sqrt{2}} = \frac{|\mathbf{0}\rangle + |\mathbf{3}\rangle}{\sqrt{2}} \tag{3.5}$$

midiendo solo un qubit puedo saber el estado de todo el sistema. Por ejemplo, suponiendo que se mide el segundo qubit y da como resultado 1, entonces como $a_{\mathbf{2}} = 0$, la probabilidad del sistema de encontrarse en el estado $|10\rangle = |\mathbf{2}\rangle$ es nula, por lo que el estado post medición del sistema es $|\psi\rangle = |11\rangle = |\mathbf{3}\rangle$. Pero entonces, a pesar de no haber medido el primer qubit, el mismo se encuentra inequívocamente en el estado excitado, ya que al colapsar el sistema a los estados con segundo qubit excitado, la probabilidad de medir $|00\rangle = |\mathbf{0}\rangle$ pasa a ser nula. Este y los demás Estados de Bell forman una base del sistema de dos qubits, y en particular, la base con *máximo entrelazamiento*, en un sentido que se explicará en la sección 3.1.4.

La generalización a más qubits es trivial: suponiendo se tiene un sistema de n qubit, su estado puede describirse como

$$|\psi\rangle = \sum_{x=0}^{2^n-1} a_x |x\rangle \tag{3.6}$$

donde los coeficientes cumplen la condición $1 = \sum_{x=0}^{2^n-1} |\alpha_x|^2$. Se puede observar el aumento exponencial en la cantidad de información oculta en el sistema con el número de qubits: todos estos 2^n coeficientes α_x evolucionan junto con el sistema a medida que este es manipulado, por lo que esta información puede ser *procesada*. El objetivo es entonces lograr extraer la mayor cantidad de información útil de esta información oculta.

3.1.3. Operadores y Compuertas Cuánticas

Con el objetivo de modificar el estado de un qubit, se aplican sobre él *operadores*. Aplicar un operador U a un vector $|\psi\rangle$ denota la transformación $|\psi\rangle \to U|\psi\rangle = |\psi'\rangle \in \mathbb{C}^2$ donde $|\psi'| = 1$. Como la aplicación de U es una operación que preserva la norma, se tiene que son operadores unitarios, es decir, se cumple que $|U|^2 = UU^\dagger = 1$, con $U^\dagger$ el operador adjunto a U, lo que también implica que se preserva el producto interno: $\langle U\chi, U\psi\rangle = \langle \chi, U^\dagger U\psi\rangle = \langle \chi, \psi\rangle$. Los operadores pueden representarse mediante matrices, también unitarias, con coeficientes complejos. En analogía con la computación clásica, estas matrices se denominan compuertas cuánticas y modifican el estado del sistema, es decir, los coeficientes α y β.

Un operador de un qubit general puede escribirse como una matriz

$$U = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \tag{3.7}$$

y su acción sobre un estado arbitrario de un qubit es

$$U|\psi\rangle = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} a\alpha + b\beta \\ c\alpha + d\beta \end{bmatrix} \tag{3.8}$$

Otra forma de representar estos operadores, en vez de matrices, es la notación *producto externo*

$$U = \begin{bmatrix} a & b \\ c & d \end{bmatrix} = a|0\rangle\langle 0| + b|0\rangle\langle 1| + c|1\rangle\langle 0| + d|1\rangle\langle 1| \tag{3.9}$$

en la cuál la fila i-ésima corresponde a $|i\rangle$ y análogamente, la columna j-ésima se representa mediante $\langle j|$. Por lo tanto,

$$\begin{aligned} U|\psi\rangle &= (a|0\rangle\langle 0| + b|0\rangle\langle 1| + c|1\rangle\langle 0| + d|1\rangle\langle 1|)(\alpha|0\rangle + \beta|1\rangle) \\ &= a\alpha|0\rangle\langle 0|0\rangle + b\beta|0\rangle\langle 1|1\rangle + c\alpha|1\rangle\langle 0|0\rangle + d\beta|1\rangle\langle 1|1\rangle \\ &= a\alpha|0\rangle + b\beta|0\rangle + c\alpha|1\rangle + d\beta|1\rangle \end{aligned} \tag{3.10}$$

Esta notación ofrece la ventaja de la facilidad para representar operadores n-dimensionales de la misma forma que se representan los vectores de estado y además, permite ignorar todos los coeficientes iguales a 0 lo cuál es muy útil al trabajar con matrices ralas o *sparse*. Por ejemplo, una matriz identidad $n \times n$ se describe simplemente como $I_n = \sum_{x=1}^{n} |x\rangle \langle x|$.

Algunas de las compuertas sobre un qubit mas importantes son la de Hadamard (3.11), la de negación o NOT (3.12) y la de rotación en una fase ϕ o $R(\phi)$ (3.13).

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} = \frac{1}{\sqrt{2}}(|0\rangle \langle 0| + |0\rangle \langle 1| + |1\rangle \langle 0| - |1\rangle \langle 1|) \tag{3.11}$$

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} = |0\rangle \langle 1| + |1\rangle \langle 0| \tag{3.12}$$

$$R(\phi) = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\phi} \end{bmatrix} = |0\rangle \langle 0| + e^{i\phi} |1\rangle \langle 1| \tag{3.13}$$

Otro operador o compuerta importante es la Z, que se corresponde a una rotación en una fase $\phi = \pi$

$$Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} = |0\rangle \langle 0| - |1\rangle \langle 1| \tag{3.14}$$

3.1.4. Producto Tensorial

¿Como actúa un operador en un sistema de varios qubits? ¿Como se aplica un operador a *un solo qubit* de un sistema compuesto? La respuesta a estas preguntas se haya en el *Producto Tensorial*. El producto tensorial es una forma de construir espacio vectoriales a partir de espacios vectoriales mas pequeños.

Supongamos que V y W son dos espacios vectoriales. Entonces se define el espacio producto $V \otimes W$ como el espacio generado por todos los vectores $|v\rangle \otimes |w\rangle$ los cuáles cumplen que $|v\rangle \in V$ y $|w\rangle \in W$. Sea $\{|i\rangle\}$ y $\{|j\rangle\}$ bases ortonormales de V y W respectivamente, entonces el conjunto $\{|i\rangle \otimes |j\rangle\}$ o, en notación abreviada, $\{|ij\rangle\}$ forma una base ortonormal del espacio producto. Observemos que en el caso particular en que V y W son sistemas de dos niveles, entonces $\{|i\rangle \otimes |j\rangle\} = \{|00\rangle, |01\rangle, |10\rangle, |11\rangle\} = \{|\mathbf{0}\rangle, |\mathbf{1}\rangle, |\mathbf{2}\rangle, |\mathbf{3}\rangle\}$, como vimos en la sección 3.1.2.

El producto tensorial es distributivo y cuantitativo con respecto a la suma de vectores y la multiplicación por un escalar. Entonces, un estado general del espacio producto se representa como

$$|\psi\rangle = \sum_{i,j} \alpha_{ij} |i, j\rangle \tag{3.15}$$

Sean $|v\rangle = \sum_i v_i |i\rangle \in V$ y $|w\rangle = \sum_j w_i |j\rangle \in W$, entonces el estado producto $|v\rangle \otimes |w\rangle \in V \otimes W$ es

$$|v\rangle \otimes |w\rangle = \big(\sum_i v_i |i\rangle\big) \otimes \big(\sum_j w_j |j\rangle\big) = \sum_{i,j} (v_i |i\rangle \otimes w_j |j\rangle) \tag{3.16}$$

Observemos que si bien el conjunto $\{|v\rangle \otimes |w\rangle\} \subset V \otimes W$, lo opuesto no se cumple, es decir, existen estados en $V \otimes W$ que no pueden ser representados como productos tensoriales de estados de V y W. Veamos un ejemplo: Consideremos un sistema de dos qubits inicialmente en el estado de Bell $|\Phi^+\rangle$ (3.5). Supongamos $|\Phi^+\rangle = |v\rangle \otimes |w\rangle$. Entonces

$$\begin{aligned} |\Phi^+\rangle &= \frac{|00\rangle + |11\rangle}{\sqrt{2}} = \frac{|\mathbf{0}\rangle + |\mathbf{3}\rangle}{\sqrt{2}} \\ &= |v\rangle \otimes |w\rangle = (v_0 |0\rangle + v_1 |1\rangle) \otimes (w_0 |0\rangle + w_1 |1\rangle) \\ &= v_0 |0\rangle \otimes w_0 |0\rangle + v_0 |0\rangle \otimes w_1 |1\rangle + v_1 |1\rangle \otimes w_0 |0\rangle + v_1 |1\rangle \otimes w_1 |1\rangle \\ &= v_0 w_0 |00\rangle + v_0 w_1 |01\rangle + v_1 w_0 |10\rangle + v_1 w_1 |11\rangle \\ &= v_0 w_0 |\mathbf{0}\rangle + v_0 w_1 |\mathbf{1}\rangle + v_1 w_0 |\mathbf{2}\rangle + v_1 w_1 |\mathbf{3}\rangle \end{aligned} \tag{3.17}$$

Esto implica las siguiente ecuaciones

$$v_0 w_0 = \frac{1}{\sqrt{2}} \quad v_0 w_1 = 0 \quad v_1 w_0 = 0 \quad v_1 w_1 = \frac{1}{\sqrt{2}} \tag{3.18}$$

Este conjunto de ecuaciones no tiene solución, por lo que $\nexists\, |v\rangle, |w\rangle \in \mathbb{C}^2$ tales que $|\Phi^+\rangle = |v\rangle \otimes |w\rangle$.

Operadores sobre varios qubits

En general, un operador que actúa sobre vectores del espacio producto $V \otimes W$ se define de la siguiente forma, en notación producto externo

$$U = \sum_{i,j,k,l} u_{ijkl} |ij\rangle \langle kl| \tag{3.19}$$

donde $\{|ij\rangle\}$ son una base de $V \otimes W$.

Si se tienen A y B operadores que actuan sobre V y W respectivamente, se puede construir un operador producto $A \otimes B$ que actúa sobre $V \otimes W$, y el mismo se define en base a la acción de A y B sobre los vectores v y w.

$$(A \otimes B)(|v\rangle \otimes |w\rangle) = A |v\rangle \otimes B |w\rangle \tag{3.20}$$

Por lo tanto,

$$A \otimes B = \sum_{i,k} (a_{ik} |i\rangle \langle k|) \otimes \sum_{j,l} (b_{jl} |j\rangle \langle l|) = \sum_{i,k} \sum_{j,l} a_{ik} b_{jl} |ij\rangle \langle kl| \tag{3.21}$$

Una forma mas sencilla de visualizar los coeficiente de $A \otimes B$ es mediante el *producto de Kronecker* de matrices. Sean A y B matrices $m \times n$ y $p \times q$ respectivamente:

$$A \otimes B = \begin{bmatrix} a_{11}B & \cdots & a_{1n}B \\ \vdots & \ddots & \\ a_{m1}B & & a_{mn}B \end{bmatrix} \tag{3.22}$$

donde $a_{ij}B$ denota los bloques

$$a_{ij}B = \begin{bmatrix} a_{ij}b_{11} & \cdots & a_{ij}b_{1q} \\ \vdots & \ddots & \\ a_{ij}b_{p1} & & a_{ij}b_{pq} \end{bmatrix} \tag{3.23}$$

A modo de ejemplo, apliquemos la compuerta de Hadamard a cada qubit de un sistema de dos qubits, es decir, el operador $H \otimes H$ o $H^{\otimes 2}$.

$$H^{\otimes 2} = \begin{bmatrix} H & H \\ H & -H \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix} \tag{3.24}$$

Entonces

$$H^{\otimes 2} |\psi\rangle = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix} \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{bmatrix} = \begin{bmatrix} \alpha_0 + \alpha_1 + \alpha_2 + \alpha_3 \\ \alpha_0 + \alpha_1 + \alpha_2 - \alpha_3 \\ \alpha_0 + \alpha_1 - \alpha_0 - \alpha_3 \\ \alpha_0 - \alpha_1 - \alpha_2 + \alpha_3 \end{bmatrix} \tag{3.25}$$

Puede demostrarse que, en general

$$H^{\otimes n} = \frac{1}{\sqrt{2^n}} \sum_{x,y} (-1)^{xy} |x\rangle \langle y| \tag{3.26}$$

Una clase de operadores muy importantes son los llamados *operadores controlados*. En estos, solo se aplica cierto operador U en el caso en que el o los qubits de control se encuentren en el estado $|1\rangle$. Por ejemplo, los operadores $CNOT$ y $CR(\phi)$, que son las compuertas de negación y rotación en un fase ϕ controladas, tienen la siguiente forma para el caso en que el se tiene un solo qubit de control y el operador se aplica en el segundo qubit:

$$CNOT = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} = I_2 \oplus X_2 \tag{3.27}$$

$$CR(\phi) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e^{i\phi} \end{bmatrix} = I_2 \oplus R(\phi) \tag{3.28}$$

Del mismo modo que con los vectores, no todos los operadores que operan en el espacio $V \otimes W$ pueden escribirse como producto de operadores sobre V y W. Sin embargo, estos últimos son de mayor interés en lo que concierne a la computación cuántica. Si se quiere aplicar cierto operador A a solo una parte del vector producto tensorial $|v\rangle \otimes |w\rangle$, digamos $|v\rangle$, se puede extender el mismo a todo el espacio $V \otimes W$ mediante el producto tensorial:

$$A(|v\rangle \otimes |w\rangle) = (A \otimes I)(|v\rangle \otimes |w\rangle) = A\,|v\rangle \otimes |w\rangle \tag{3.29}$$

Por ejemplo, suponiendo que se quiere aplicar una compuerta de Hadamard al primer qubit de un sistema de dos qubits, el operador a aplicar sobre el vector de estado del sistema es $H \otimes I_2$. Usando el producto de Kronecker (3.22), los coeficientes de $H \otimes I_2$ son

$$H \otimes I_2 = \begin{bmatrix} I_2 & I_2 \\ I_2 & -I_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix} \tag{3.30}$$

Entonces

$$(H \otimes I_2)\,|\psi\rangle = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix} \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{bmatrix} = \begin{bmatrix} \alpha_0 + \alpha_2 \\ \alpha_1 + \alpha_3 \\ \alpha_0 - \alpha_2 \\ \alpha_1 - \alpha_3 \end{bmatrix} \tag{3.31}$$

Por otro lado, si lo que quiero es aplicar la compuerta de Hadamard solo al segundo qubit, el operador es $I_2 \otimes H$ y tiene coeficientes

$$I_2 \otimes H = \begin{bmatrix} H & 0 \\ 0 & H \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & -1 \end{bmatrix} \tag{3.32}$$

Entonces

$$(I_2 \otimes H)\,|\psi\rangle = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & -1 \end{bmatrix} \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{bmatrix} = \begin{bmatrix} \alpha_0 + \alpha_1 \\ \alpha_1 - \alpha_1 \\ \alpha_2 + \alpha_3 \\ \alpha_2 - \alpha_3 \end{bmatrix} \tag{3.33}$$

Como se puede observar, el producto tensorial es no conmutativo.

Suma directa

Obsérvese tambien la particular forma del operador en la ecuación (3.32). Esta matriz es una matriz *diagonal por bloques*, es decir, esta compuesta por sucesivas matrices de dimensión menor sobre la diagonal y es nula fuera de estas. Es más, en el caso de producto tensorial a izquierda por una identidad, estos bloques son la matriz a la derecha del producto. Esto puede sistematizarse mediante la notación *suma directa*.

Se define la suma directa de dos operadores $A \oplus B$ como la matriz diagonal por bloques

$$A \oplus B = \begin{bmatrix} A & 0 \\ 0 & B \end{bmatrix} \tag{3.34}$$

Por lo tanto, sea A una matriz $m \times n$ y B una matriz $p \times q$, $A \oplus B$ es una matriz $(m+p) \times (n+q)$. Volviendo al caso de la compuerta de Hadamard (3.32), observemos entonces que $I_2 \otimes H = H \oplus H$. En general, $I_n \otimes A = \bigoplus^n A$ para cualquier n y A.

Veamos algunas propiedades de la suma directa. Primero, $A \oplus B \neq B \oplus A$, por lo que la suma directa es no conmutativa. Ahora bien, con respecto a la suma directa se tiene que el producto tensorial es distributivo a derecha mas no a izquierda. Por simplicidad supongamos A, B y C son matrices $m \times n$

$$A \otimes (B \oplus C) = A \otimes \begin{bmatrix} B & 0 \\ 0 & C \end{bmatrix} = \begin{bmatrix} a_{11}B \oplus C & \cdots & a_{1n}B \oplus C \\ \vdots & \ddots & \\ a_{m1}B \oplus C & & a_{mn}B \oplus C \end{bmatrix} \neq A \otimes B \oplus A \otimes C = \begin{bmatrix} A \otimes B & 0 \\ 0 & A \otimes C \end{bmatrix} \tag{3.35}$$

$$(B \oplus C) \otimes A = \begin{bmatrix} B & 0 \\ 0 & C \end{bmatrix} \otimes A = \begin{bmatrix} b_{11}A & \cdots & b_{1n}A & 0 & \cdots & 0 \\ \vdots & \ddots & & \vdots & & \\ b_{m1}A & \cdots & b_{mn}A & 0 & \cdots & 0 \\ 0 & \cdots & 0 & c_{11}A & \cdots & c_{1n}A \\ \vdots & & & \vdots & \ddots & \\ 0 & \cdots & 0 & c_{m1}A & \cdots & c_{1m}A \end{bmatrix} = \begin{bmatrix} B \otimes A & 0 \\ 0 & C \otimes A \end{bmatrix} = B \otimes A \oplus C \otimes A \tag{3.36}$$

3.1.5. Mediciones

Los postulados de la Mecánica Cuántica establecen que las mediciones pueden ser descriptas mediante un *observable* M. Dicho observable es un operador hermitiano en el espacio de estados del sistema, y tiene una descomposición espectral dada por

$$M = \sum_m m P_m \tag{3.37}$$

donde P_m es el proyector al subespacio expandido por los autovectores del autovalor m, es decir, el autoespacio $m \in V$, donde V es el espacio vectorial del sistema compuesto de n qubits.

$$P_m = \sum_j |j\rangle \langle j| \tag{3.38}$$

con $\{|j\rangle\}$ una base del autoespacio m.

Luego de ser medido y obtener un autovalor m, el vector de estado es proyectado al autoespacio de m, y la probabilidad de medir dicho autovalor esta dada por el valor de expectación de su proyector justo antes de la medición. Es decir

$$P(m) = \langle\psi| P_m |\psi\rangle \tag{3.39}$$

Como P_m no son operadores unitarios, el estado $|\psi'\rangle$ es renormalizado dividiéndolo por $\sqrt{P(m)}$.

$$|\psi'\rangle = \frac{P_m |\psi\rangle}{\sqrt{P(m)}} \tag{3.40}$$

Se tiene que los proyectores satisfacen la condición de completitud, es decir, que expanden todo el espacio de estados: $\sum_m P_m = I_z$ con z la dimensión del espacio de estados, y además, son ortogonales entre sí.

Por ejemplo, si queremos medir el estado del sistema, los autovalores de M son el número del vector medido y los proyectores simplemente $P_m = |m\rangle \langle m|$. Entonces el observable M es

$$M = \sum_{m=0}^{2^z-1} m |m\rangle \langle m| \tag{3.41}$$

Sin embargo, también pueden medirse solo partes del sistema. Basándonos en lo discutido en la sección 3.1.4, si quiero medir el qubit i-ésimo, solo tengo que construir M para el espacio del qubit i-ésimo, o sea, $M = \sum_m mP_m$ con $P_0 = |0\rangle \langle 0|$ y $P_1 = |1\rangle \langle 1|$. Entonces, la probabilidad y el estado post medición, según el resultado de esta sea 0 o 1 es, respectivamente

$$P(0) = \langle\psi| \left(I_{2^{i-1}} \otimes P_0 \otimes I_{2^{z-i}}\right) |\psi\rangle \tag{3.42}$$

$$|\psi'\rangle = \frac{P_0 |\psi\rangle}{\sqrt{P(0)}} \tag{3.43}$$

$$P(1) = \langle\psi| \left(I_{2^{i-1}} \otimes P_1 \otimes I_{2^{z-i}}\right) |\psi\rangle = 1 - P_0 \tag{3.44}$$

$$|\psi'\rangle = \frac{P_1 |\psi\rangle}{\sqrt{P(1)}} \tag{3.45}$$

El estado del qubit i colapsa al estado medido, pero en el resto de qubits se mantiene la superposición.

3.1.6. Circuitos Cuánticos

Supongamos que se quieren aplicar, en serie, dos operadores sobre el primer qubit, A y B, mientras que el segundo qubit se lo deja inalterado. Luego, se aplica un tercer operador C sobre el segundo qubit. En notación estándar de operadores:

$$(I_2 \otimes C)(B \otimes I_2)(A \otimes I_2)\,|\psi\rangle \tag{3.46}$$

Este notación se torna oscura a medida que aumentan la cantidad de qubits y la cantidad de operadores a aplicar.

Los *circuitos cuánticos* son un modelo de computación cuántica, basado en los circuitos clásicos, y son especialmente útiles a la hora de estudiar los algoritmos, por su intuitiva interpretación gráfica. En ellos, cada qubit se representa mediante una linea horizontal, a la que se le llaman cables o *wires*, sobre la cuál se colocan cajas u otros símbolos que representan las distintas compuertas. Los circuitos se leen de izquierda a derecha, es decir, una compuerta se aplica antes que otra que se encuentre mas a la derecha. Esto es particularmente útil cuando en un sistema de muchos qubits, se quiere aplicar un operador solamente a uno o pocos de ellos, ya que en esta notación no es necesario colocar las compuertas correspondientes a las identidades, debido a que estas no cambian el estado del qubit sobre el que están aplicadas. Por ejemplo, el caso anterior puede describirse usando circuitos como

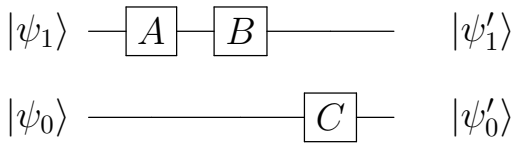


Figura 3.2: Ejemplo de circuito cuántico

Aquí, $|\psi_1\rangle$ representa el primer qubit y $|\psi_0\rangle$ el segundo, es decir $|\psi\rangle = |\psi_1\rangle \otimes |\psi_0\rangle$. Las compuertas que actúan sobre mas de un qubit se representa con cajas que abarcan varios cables. Por ejemplo, la acción de aplicar U sobre los dos qubits se representa como

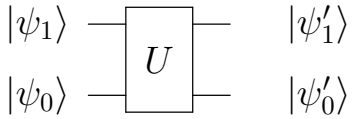


Figura 3.3: Compuerta U sobre dos qubits

Reversibilidad de los circuitos cuánticos Y Teorema de No Clonación

Al ser las compuertas cuánticas operadores unitarios, se tiene que estos son mapas biyectivos dentro del espacio de estados del sistema. Esto significa que a cada estado de entrada le corresponde un y sólo un estado de salida, y viceversa. Por lo tanto, un circuito cuántico puede ser recorrido de derecha a izquierda recuperando los estados iniciales, eso si, reemplazando cada operador por su adjunto. A este tipo de cómputos se les denomina *reversible*. Lo primero que podemos observar de un circuito reversible es que los cables correspondientes a cada bit o qubit son lineas rectas que van de principio a fin del algoritmo, ya que la reversibilidad implica que cada operador tiene la misma cantidad de entradas que de salidas. Esta es una gran diferencia a la hora de implementar la mayor parte de los circuitos clásicos, ya que en estos la reversibilidad no es un requisito. Hay otra limitación propia de la mecánica cuántica que impone más limitaciones a los circuitos cuánticos en comparación con los clásicos:

Theorem 1 (Teorema de No Clonación)**.** *No existe ningún proceso en la Mecánica Cuántica que tome un vector de estado arbitrario $|\psi\rangle$ como entrada y produzca como salida dos vectores de estado separados idénticos al estado original.*

Veamos un caso práctico de sus consecuencias. La operación clásica $FANOUT$ puede a su vez ser implementada mediante una operación lógica controlada $CNOT$, que puede definirse análogamente a la compuerta cuántica. Esta compuerta invierte el estado de un bit si y solo si un segundo bit de control vale 1. Luego, ante una entrada $x0$, la salida siempre es xx para cualquier estado posible del bit de control, $x = 0, 1$. Ahora intentemos de replicar esto con qubits: en este caso, el estado a copiar, el qubit de control, es $|\psi\rangle = a|0\rangle + b|1\rangle$ con a, b arbitrarios.Entonces, el vector de entrada es

$$(a|0\rangle + b|1\rangle) \otimes |0\rangle = a|00\rangle + b|10\rangle \tag{3.47}$$

Si ahora aplicamos la compuerta $CNOT$, obtenemos

$$\begin{aligned} CNOT|\psi\rangle|1\rangle = CNOT(a|00\rangle + b|10\rangle) = a|00\rangle + b|11\rangle \\ \neq |\psi\rangle|\psi\rangle = a^2|00\rangle + ab|01\rangle + ba|10\rangle + b^2|11\rangle \end{aligned} \tag{3.48}$$

La igualdad solo vale si $ab = 0$, lo cuál a su vez, solo vale si $a, b = 0, 1$ o $a, b = 1, 0$. Este caso se corresponde a un bit clásico, lo que esta en concordancia con que las leyes de la física clásica son, en última instancia, un caso límite particular de las leyes de la mecánica cuántica.

Sin embargo, se demostrara en el próximo capítulo que cualquier computo no reversible puede ser transformado en reversible, a costa de a lo mas la utilización de qubits extra de ayuda al computo, los bits ancilla.

3.2. Compuertas Fundamentales

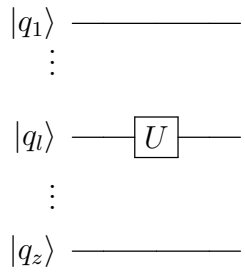
Como vimos en la sección anterior, los operadores cuánticos son operadores hermitianos sobre un espacio de Hilbert complejo que a su vez pueden representarse como matrices, y por lo tanto responden a las reglas del Álgebra lineal. De esta manera, aplicar un operador sobre un vector de estado es simplemente multiplicar a izquierda la matriz, mientras que la matriz de dos operadores que actúen de forma sucesiva se obtiene multiplicando las matrices de cada uno de ellos. A pesar de la simplicidad que el Álgebra de matrices confiere, existe un problema práctico en la escalabilidad de las operaciones con matrices. En un sistema de n qubits, el número de coeficientes necesarios para definir un vector de estado es 2^n, pero las matrices de los operadores que actúan sobre el tiene 2^{2n} coeficientes. Luego, para obtener cada uno de los coeficientes del vector resultante son necesarias 2^n multiplicaciones y $2^n - 1$ sumas, por lo que el total de operaciones necesarias al aplicar un operador son $\Theta(2^{2n})$ operaciones.

Sin embargo, como se verá en esta sección, los coeficientes de las matrices son en su mayoría nulos, por lo que se las llama matrices *sparse* o ralas. En particular, son o bien diagonales o bien tridiagonales, es decir que todos sus coeficientes no nulos están en tres columnas. Usando la notación producto externo, es directo deducir expresiones algebraicas que solo tengan en cuenta los coeficientes no nulos, reduciendo el numero de operaciones por elemento del vector de estado a a lo sumo dos multiplicaciones y una suma, y en total $O(2^n)$.

3.2.1. Compuertas sobre un qubit o dos qubits adyacentes

Compuertas sobre un solo qubit

Supongamos que se quiere aplicar un operador U a un qubit en particular, llamémoslo el l-ésimo, de un total de z qubits. En el modelo de circuitos, esta operación puede denotarse como muestra la siguiente figura:



(3.49)

Sea

$$U = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \tag{3.50}$$

En notación producto externo:

$$U = a\,|0\rangle\langle 0| + b\,|0\rangle\langle 1| + c\,|1\rangle\langle 0| + d\,|1\rangle\langle 1| \tag{3.51}$$

Entonces, la operación a realizar sobre el vector producto tensorial $|\psi\rangle$ es

$$\begin{aligned} O\,|\psi\rangle &= (I_{2^{l-1}} \otimes U \otimes I_{2^{z-l}})\,|\psi\rangle \\ &= (I_{2^{l-1}} \otimes U \otimes I_{2^{z-l}})\,|\psi\rangle \\ &= (I_{2^{l-1}}\,|q_1 \cdots q_{l-1}\rangle \otimes U\,|l\rangle \otimes I_{2^{z-l}}\,|q_{z-l+1} \cdots q_z\rangle) \\ &= |q_1 \cdots q_{l-1}\rangle \otimes U\,|l\rangle \otimes |q_{z-l+1} \cdots q_z\rangle \end{aligned} \tag{3.52}$$

Sabemos que el efecto de multiplicar un operador U a izquierda por una identidad es

$$I_2 \otimes U = U \oplus U = \begin{bmatrix} U & 0 \\ 0 & U \end{bmatrix} \tag{3.53}$$

Y generalizando,

$$I_{2^L} \otimes U = U \oplus \cdots \oplus U = \begin{bmatrix} U & \cdots & O \\ \vdots & \ddots & \\ 0 & & U \end{bmatrix} \tag{3.54}$$

Es decir, la suma directa 2^L veces de U.

Por otro lado, multiplicar U a derecha por una identidad da como resultado

$$U \otimes I_2 = a\,|0\rangle\langle 0| \otimes I_2 + b\,|0\rangle\langle 1| \otimes I_2 + c\,|1\rangle\langle 0| \otimes I_2 + d\,|1\rangle\langle 1| \otimes I_2$$

$$U \otimes I_2 = \begin{bmatrix} aI_2 & bI_2 \\ cI_2 & dI_2 \end{bmatrix} \tag{3.55}$$

Y generalizando,

$$U \otimes I_{2^Z} = a\,|0\rangle\langle 0| \otimes I_{2^Z} + b\,|0\rangle\langle 1| \otimes I_{2^Z} + c\,|1\rangle\langle 0| \otimes I_{2^Z} + d\,|1\rangle\langle 1| \otimes I_{2^Z}$$

$$U \otimes I_{2^Z} = \begin{bmatrix} aI_{2^Z} & bI_{2^Z} \\ cI_{2^Z} & dI_{2^Z} \end{bmatrix} \tag{3.56}$$

Luego,

$$I_{2^L} \otimes U \otimes I_{2^Z} = I_{2^L} \otimes (a\,|0\rangle\langle 0| \otimes I_{2^Z} + b\,|0\rangle\langle 1| \otimes I_{2^Z} + c\,|1\rangle\langle 0| \otimes I_{2^Z} + d\,|1\rangle\langle 1| \otimes I_{2^Z})$$

$$I_{2^L} \otimes U \otimes I_{2^Z} = \begin{bmatrix} aI_{2^Z} & bI_{2^Z} & \cdots & & 0 \\ cI_{2^Z} & dI_{2^Z} & & & \\ \vdots & & \ddots & & \\ & & & aI_{2^Z} & bI_{2^Z} \\ 0 & & & cI_{2^Z} & dI_{2^Z} \end{bmatrix} \tag{3.57}$$

Es decir, la suma directa 2^L veces de $U \otimes I_{2^Z}$.

Notemos que esta matriz, de dimensión 2^{Z+L+1}, es tridiagonal

$$I_{2^L} \otimes U \otimes I_{2^Z} = I_{2^L} \otimes \begin{bmatrix} a & \cdots & 0 & b & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdot & a & 0 & \cdots & b \\ c & \cdots & 0 & d & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & c & 0 & \cdots & d \end{bmatrix} \tag{3.58}$$

En notación producto externo:

$$\begin{aligned} I_{2^L} \otimes U \otimes I_{2^Z} = & a(|0\rangle\langle 0| + \cdots + |2^Z - 1\rangle\langle 2^Z - 1|) \\ & + b(|0\rangle\langle 2^Z - 1| + \cdots + |2^Z - 1\rangle\langle 2^{Z+1} - 1|) \\ & + c(|2^Z - 1\rangle\langle 0| + \cdots + |2^{Z+1} - 1\rangle\langle 2^Z - 1|) \\ & + d(|2^Z - 1\rangle\langle 2^Z - 1| + \cdots + |2^{Z+1} - 1\rangle\langle 2^{Z+1} - 1|) + \ldots \end{aligned} \tag{3.59}$$

$$
\begin{aligned}
&= a \sum_{i=0}^{2^Z-1} (|i\rangle \langle i| + |i+2^{Z+1}\rangle \langle i+2^{Z+1}| + \cdots + |i+2^{Z+L}-2^Z\rangle \langle i+2^{Z+L}-2^Z|) \\
&+ b \sum_{i=0}^{2^Z-1} (|i\rangle \langle i+2^Z| + |i+2^Z\rangle \langle i+2^{Z+1}| + \cdots + |i+2^{Z+L}-2^Z\rangle \langle i+2^{Z+L+1}-2^Z|) \\
&+ c \sum_{i=0}^{2^Z-1} (|i+2^Z\rangle \langle i| + |i+2^{Z+1}\rangle \langle i+2^Z| + \cdots + |i+2^{Z+L+1}-2^Z\rangle \langle i+2^{Z+L}-2^Z|) \\
&+ d \sum_{i=0}^{2^Z-1} (|i+2^Z\rangle \langle i+2^Z| + |i+2^{Z+2}\rangle \langle i+2^{Z+2}| + \cdots + |i+2^{Z+L+1}-2^Z\rangle \langle i+2^{Z+L+1}-2^Z|) \\
&= a \sum_{k=0}^{2^L-1} \sum_{i=0}^{2^Z-1} (|i+2k2^Z\rangle \langle i+2k2^Z|) + b \sum_{k=0}^{2^L-1} \sum_{i=0}^{2^Z-1} (|i+2k2^Z\rangle \langle i+(2k+1)2^Z|) \\
&+ c \sum_{k=0}^{2^L-1} \sum_{i=0}^{2^Z-1} (|i+(2k+1)2^Z\rangle \langle i+2k2^Z|) + d \sum_{k=0}^{2^L-1} \sum_{i=0}^{2^Z-1} (|i+(2k+1)2^Z\rangle \langle i+(2k+1)2^Z|)
\end{aligned}
\tag{3.60}
$$

Finalmente,

$$
\begin{aligned}
I_{2^L} \otimes U \otimes I_{2^Z} = \sum_{k=0}^{2^L-1} \sum_{i=0}^{2^Z-1} &(|i+2k2^Z\rangle (a(\langle i+2k2^Z| + b \langle i+(2k+1)2^Z|) \\
&+ |i+(2k+1)2^Z\rangle ((c(\langle i+2k2^Z| + d \langle i+(2k+1)2^Z|))
\end{aligned}
\tag{3.61}
$$

Volviendo a la situación inicial, de aplicar U al operador l-ésimo, de un total de z qubits, en la ecuación (3.61) hay que reemplazar $L \to l-1$ y $Z \to z-l$. Por lo tanto, sea $A = 2^L = 2^{l-1}$ y $B = 2^Z = 2^{z-l}$

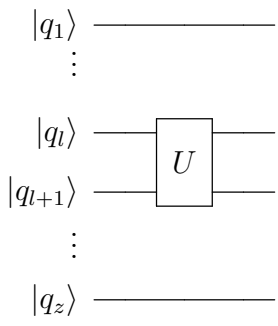
$$
\begin{aligned}
I_{2^{l-1}} \otimes U \otimes I_{2^{z-l}} = \sum_{k=0}^{2^{l-1}-1} \sum_{i=0}^{2^{z-l}-1} &(|2k2^{z-l}+i\rangle (a \langle 2k2^{z-l}+i| + b \langle (2k+1)2^{z-l}+i|) \\
&+ |(2k+1)2^{z-l}+i\rangle (c \langle 2k2^{z-l}+i| + d \langle (2k+1)2^{z-l}+i|))
\end{aligned}
\tag{3.62}
$$

Las compuertas de Hadamard (3.11), la de negación o NOT (3.12) y la de rotación en una fase ϕ o $R_(\phi)$ (3.13), constituyen un conjunto *fundamental*, es decir, cualquier compuerta sobre un qubit puede representarse a partir de ellas. Dichas compuertas pueden construirse a partir del desarrollo anterior tomando los coeficientes a, b, c y d que correspondan en cada caso.

Compuertas sobre dos qubits adyacentes

Para el caso de que U es un operador que actúa en 2 qubits adyacentes, la generalización es inmediata: Quiero aplicar un operador U a los qubits l y $l+1$-ésimos, de un total de z qubits. En

notación de circuito cuántico:



(3.63)

Sea

$$U = \sum_{l,j=0}^{3} a_{lj} \left|l\right\rangle \left\langle j\right| \tag{3.64}$$

Entonces

$$I_{2^{l-1}} \otimes U \otimes I_{2^{z-l-1}} = \sum_{k=0}^{2^{l-1}-1} \sum_{i=0}^{2^{z-l-1}-1} \sum_{u,j=0}^{3} (a_{uj} \left|(4k+u)2^{z-l-1}+i\right\rangle \left\langle(4k+j)2^{z-l-1}+i\right|) \tag{3.65}$$

De forma análoga, pueden considerarse operadores que actúen sobre mas de 2 qubits adyacentes; la generalización del circuito cuántico es trivial. Para aplicar un operador sobre dos qubits adyacentes concreto, solo hay que reemplazar los coeficientes a_{lj} según corresponda. Por ejemplo, si quiero aplicar la compuerta CNOT:

$$U = \left|\mathbf{0}\right\rangle \left\langle\mathbf{0}\right| + \left|\mathbf{1}\right\rangle \left\langle\mathbf{1}\right| + \left|\mathbf{3}\right\rangle \left\langle\mathbf{2}\right| + \left|\mathbf{2}\right\rangle \left\langle\mathbf{3}\right| \tag{3.66}$$

Y si quiero aplicar CR(ϕ)

$$U = \left|\mathbf{0}\right\rangle \left\langle\mathbf{0}\right| + \left|\mathbf{1}\right\rangle \left\langle\mathbf{1}\right| + \left|\mathbf{2}\right\rangle \left\langle\mathbf{2}\right| + e^{i\phi} \left|\mathbf{3}\right\rangle \left\langle\mathbf{3}\right| \tag{3.67}$$

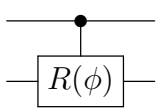
3.2.2. Compuertas CNOT y CR(ϕ) sobre qubits no adyacentes

Ahora, el siguiente paso son operadores que actúan sobre 2 qubits pero no adyacentes. La forma en que estos operadores actúan sobre el vector producto tensorial no es trivial como en los casos anteriores, ya que no pueden descomponerse simplemente en productos tensoriales de operadores e identidades.

En general, en el modelo de circuito cuántico se denotan los qubits de control mediante un punto, mientras que los qubits que se niegan mediante un signo $\oplus$ (3.68). Por su parte, los qubits que se rotan se denotan mediante $R(\phi)$ (3.69):

CNOT =  (3.68)

$$CR(\phi) = \tag{3.69}$$



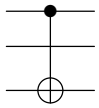
Si ahora consideramos que los operadores actúan sobre dos qubits no adyacentes separados por un qubit, tenemos lo siguiente:

$$\text{CNOT} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} = I_4 \oplus X_2 \oplus X_2 = I_4 \oplus I_2 \otimes X \tag{3.70}$$

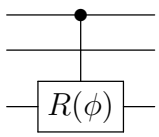
$$CR(\phi) = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & e^{i\phi} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & e^{i\phi} \end{bmatrix} = I_4 \oplus R(\phi) \oplus R(\phi) = I_4 \oplus I_2 \otimes R(\phi) \tag{3.71}$$

Y en notación circuito cuántico:

$$\text{CNOT} = \tag{3.72}$$



$$CR(\phi) = \tag{3.73}$$



En general, si quiero actuar sobre el L-ésimo qubit, siendo el primero el de control:

$$\text{CNOT}_{1L} = I_{2^{L-1}} \oplus I_{2^{L-2}} \otimes X \tag{3.74}$$

$$CR(\phi)_{1L} = I_{2^{L-1}} \oplus I_{2^{L-2}} \otimes R(\phi) \tag{3.75}$$

En notación producto externo

$$\mathrm{CNOT}_{1L} = \sum_{i=0}^{2^{L-1}-1} |i\rangle\langle i| + \sum_{i=0}^{2^{L-2}-1} \left(|2(i+2^{L-2})\rangle\langle 2(i+2^{L-2})+1| + |2(i+2^{L-2})+1\rangle\langle 2(i+2^{L-2})|\right) \tag{3.76}$$

$$CR(\phi)_{1L} = \sum_{i=0}^{2^{L-1}-1} |i\rangle\langle i| + \sum_{i=0}^{2^{L-2}-1} \left(|2(i+2^{L-2})\rangle\langle 2(i+2^{L-2})| + e^{i\phi}\,|2(i+2^{L-2})+1\rangle\langle 2(i+2^{L-2})+1|\right) \tag{3.77}$$

El siguiente objetivo es aplicar las compuertas CNOT y CR(ϕ) sobre el vector producto tensorial completo de z qubits. Como ya vimos antes, la acción de multiplicar un operador a derecha por I_{2^N} es la suma directa 2^N veces del mismo operador.

$$I_{2^N} \otimes \mathrm{CNOT}_{1L} = I_{2^N} \otimes (I_{2^{L-1}} \oplus I_{2^{L-2}} \otimes X) \tag{3.78}$$

$$I_{2^N} \otimes \mathrm{CR}(\phi)_{1L} = I_{2^N} \otimes (I_{2^{L-1}} \oplus I_{2^{L-2}} \otimes R(\phi)) \tag{3.79}$$

Si ahora multiplicamos a derecha por I_{2^Z}

$$I_{2^N} \otimes \mathrm{CNOT}_{1L} \otimes I_{2^Z} = I_{2^N} \otimes (I_{2^{L-1}} \oplus I_{2^{L-2}} \otimes X) \otimes I_{2^Z} = I_{2^N} \otimes (I_{2^{L+Z-1}} \oplus I_{2^{L-2}} \otimes X \otimes I_{2^Z}) \tag{3.80}$$

$$I_{2^N} \otimes CR(\phi)_{1L} \otimes I_{2^N} = I_{2^N} \otimes (I_{2^{L-1}} \oplus I_{2^{L-2}} \otimes R(\phi)) \otimes I_{2^N} = I_{2^N} \otimes (I_{2^{L+N-1}} \oplus I_{2^{L-2}} \otimes R(\phi) \otimes I_{2^N}) \tag{3.81}$$

Aplicar las compuertas al operador l-ésimo qubit, siendo el m-ésimo el qubit de control ($l > m$ por ahora),de un total de z qubits, es reemplazar en las ecuaciones (3.80) y (3.81) $z \to m-1$,$L \to l-m+1$ y $N \to z-l-1$, donde la compuerta entre m y l tiene dimensión 2^{l-m+1}. Por lo tanto:

$$\begin{aligned} I_{2^{m-1}} \otimes \mathrm{CNOT}_{ml} \otimes I_{2^{z-l}} &= I_{2^{m-1}} \otimes (I_{2^{l-m}} \oplus I_{2^{l-m-1}} \otimes X) \otimes I_{2^{z-l-1}} \\ &= I_{2^{m-1}} \otimes (I_{2^{z-m-1}} \oplus I_{2^{l-m-1}} \otimes X \otimes I_{2^{z-l-1}}) \end{aligned} \tag{3.82}$$

$$\begin{aligned} I_{2^{m-1}} \otimes CR(\phi)_{ml} \otimes I_{2^{z-l}} &= I_{2^{m-1}} \otimes (I_{2^{l-m}} \oplus I_{2^{l-m-1}} \otimes R(\phi)) \otimes I_{2^{z-l-1}} \\ &= I_{2^{m-1}} \otimes (I_{2^{z-m-1}} \oplus I_{2^{l-m-1}} \otimes R(\phi) \otimes I_{2^{z-l-1}}) \end{aligned} \tag{3.83}$$

Ahora buscaremos la expresión en notación producto externo de este operador. Primero, reescribo las ecuaciones (3.76) y (3.77) con $L \to l-m+1$.

$$\mathrm{CNOT}_{ml} = \sum_{i=0}^{2^{l-m}-1} |i\rangle\langle i| + \sum_{i=0}^{2^{l-m-2}-1} \left(|2(i+2^{l-m-1})\rangle\langle 2(i+2^{l-m-1})+1| + |2(i+2^{l-m-1})+1\rangle\langle 2(i+2^{l-m-1})|\right) \tag{3.84}$$

$$CR(\phi)_{ml} = \sum_{i=0}^{2^{l-m}-1} |i\rangle\langle i| + \sum_{i=0}^{2^{l-m-2}-1} (|2(i+2^{l-m-1})\rangle\langle 2(i+2^{l-m-1})| + e^{i\phi}|2(i+2^{l-m-1})+1\rangle\langle 2(i+2^{l-m-1})+1|) \tag{3.85}$$

Entonces, los operadores de las ecuaciones (3.82) y (3.83) se escriben como:

$$\begin{aligned} &I_{2^{m-1}} \otimes \mathrm{CNOT}_{ml} \otimes I_{2^{z-l}} = \\ &\sum_{k=0}^{2^{m-1}-1} \Bigg[\sum_{i=0}^{2^{z-m}-1} |i+k2^{z-m+1}\rangle\langle i+k2^{z-m+1}| + \\ &\sum_{j=0}^{2^{l-m-1}-1} \sum_{i=0}^{2^{z-l}-1} \Big(|2^{z-l}(2j+2^{l-m}+1)+i+k2^{z-m+1}\rangle\langle 2^{z-l}(2j+2^{l-m})+i+k2^{z-m+1}| \\ &+ |2^{z-l}(2j+2^{l-m})+i+k2^{z-m+1}\rangle\langle 2^{z-l}(2j+2^{l-m}+1)+i+k2^{z-m+1}| \Big) \Bigg] \end{aligned} \tag{3.86}$$

$$\begin{aligned} &I_{2^{m-1}} \otimes CR(\phi)_{ml} \otimes I_{2^{z-l}} = \\ &\sum_{k=0}^{2^{m-1}-1} \Bigg[\sum_{i=0}^{2^{z-m}-1} |i+k2^{z-m+1}\rangle\langle i+k2^{z-m+1}| + \\ &\sum_{j=0}^{2^{l-m-1}-1} \sum_{i=0}^{2^{z-l}-1} \Big(|2^{z-l}(2j+2^{l-m})+i+k2^{z-m+1}\rangle\langle 2^{z-l}(2j+2^{l-m})+i+k2^{z-m+1}| \\ &+ e^{i\phi}|2^{z-l}(2j+2^{l-m}+1)+i+k2^{z-m+1}\rangle\langle 2^{z-l}(2j+2^{l-m}+1)+i+k2^{z-m+1}| \Big) \Bigg] \end{aligned} \tag{3.87}$$

Suponiendo ahora que $l < m$, es decir, que el qubit de control está después que el que cambia, las compuertas CNOT sobre el vector producto tensorial de z qubits se transforma en:

$$\begin{aligned} &I_{2^{l-1}} \otimes \mathrm{CNOT}_{ml} \otimes I_{2^{z-m}} = \\ &\sum_{k=0}^{2^{l-1}-1} \sum_{i=0}^{2^{z-m}-1} \Bigg[\sum_{j=0}^{2^{m-l}-1} |i+2k2^{z-l}+2j2^{z-m}\rangle\langle i+2k2^{z-l}+2j2^{z-m}| \\ &\sum_{j=0}^{2^{m-l-1}-1} \Big(|i+2k2^{z-l}+(2j+1)2^{z-m}\rangle\langle i+(2k+1)2^{z-l}+(2j+1)2^{z-m}| \\ &+ |i+(2k+1)2^{z-l}+(2j+1)2^{z-m}\rangle\langle i+2k2^{z-l}+(2j+1)2^{z-m}| \Big) \Bigg] \end{aligned} \tag{3.88}$$

Por su parte, $I_{2^{l-1}} \otimes CR(\phi)_{ml} \otimes I_{2^{z-m}}$ no cambia, porque como se puede observar en la matriz (3.71), esta compuerta solo actúa sobre $|1\ldots1\rangle$, o sea, es simétrica en los qubits m y l.

3.2.3. Compuertas de Negación y Rotación con varios qubits de control

En este trabajo también se usaran compuertas de rotación y negación con doble o triple control. Según Barenco et al. en [16], cualquier compuerta U sobre un sólo qubit pero controlada por 2 qubits. llamémosla CCU, puede construirse utilizando $\Theta(8)$ compuertas CNOT y $\Theta(8)$ compuertas controladas CV, donde la compuerta de un qubit V es la raíz cuadrada de U.

CCR(ϕ)

Comenzando por CCR(ϕ), en la figura (3.89) se muestra el circuito y los operadores correspondientes a esta compuerta, para el caso en que el numero total de qubits es 3, siendo los dos primeros de control y el tercero el que se rota.

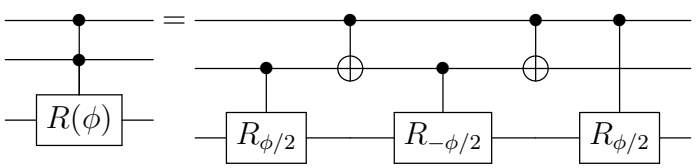
$$CCR(\phi) = \quad = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & e^{i\phi} \end{bmatrix} = I_4 \oplus I_2 \oplus R(\phi)_2 \tag{3.89}$$

Como dijimos al principio de la sección, se sabe que CCR(ϕ) puede implementarse mediante compuertas CNOT y compuertas controladas CV. Luego,

$$V^2 = R(\phi) \implies V = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e^{i\phi/2} \end{bmatrix} \tag{3.90}$$

Entonces, el circuito para implementar CCR(ϕ) es

$$= \tag{3.91}$$



CCNOT o TOFFOLI y C3NOT

En la figura (3.92) se muestra el circuito y los operadores correspondientes a la compuerta CCNOT o TOFFOLI.

$$\text{CCNOT} = \quad = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} = I_4 \oplus I_2 \oplus X \tag{3.92}$$

La CCNOT o TOFFOLI puede construirse a partir de la implementación en serie de compuertas CNOT y CROOT, como se ve en el circuito (3.93)

$$= \quad \boxed{\sqrt{X}} \quad \boxed{\sqrt{X}^\dagger} \quad \boxed{\sqrt{X}} \tag{3.93}$$

La compuerta *ROOT* o raíz cuadrada ($\sqrt{X}$) hace referencia a que dicha compuerta al cuadrado es igual a la compuerta *NOT* (3.94):

$$= \quad \boxed{\sqrt{X}} \quad \boxed{\sqrt{X}} \tag{3.94}$$

y sus coeficientes son:

$$\sqrt{X} = \frac{1}{2}\begin{bmatrix} 1+i & 1-i \\ 1-i & 1+i \end{bmatrix} \sqrt{X}^\dagger = \frac{1}{2}\begin{bmatrix} 1-i & 1+i \\ 1+i & 1-i \end{bmatrix} \tag{3.95}$$

El caso general establece que para una compuerta U controlada por k qubits, o sea CkU, esta puede ser simulada con 2^k compuertas CV tales que $V^{2^{k-1}} = U$, y $2^k - 1$ CNOT Por su parte, la compuerta C3NOT puede construirse usando $V = \sqrt[4]{X}$. La secuencia de compuertas se muestra en el circuito 3.96, mientras que en 3.97 se explicitan las compuertas $\sqrt[4]{X}$.

$$= \quad \boxed{\sqrt{X}} \quad \boxed{\sqrt[4]{X}^\dagger} \quad \boxed{\sqrt{X}} \quad \boxed{\sqrt[4]{X}^\dagger} \quad \boxed{\sqrt{X}} \quad \boxed{\sqrt[4]{X}^\dagger} \quad \boxed{\sqrt{X}} \tag{3.96}$$

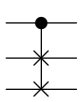
$$\sqrt[4]{X} = \frac{1}{2\sqrt{2}} \begin{bmatrix} (1+\sqrt{2})-i & (1\sqrt{2})+i \\ (1-\sqrt{2})+i & (1+\sqrt{2})-i \end{bmatrix} \quad \sqrt[4]{X}^{\dagger} = \frac{1}{2\sqrt{2}} \begin{bmatrix} (1+\sqrt{2})+i & (1\sqrt{2})-i \\ (1-\sqrt{2})-i & (1+\sqrt{2})+i \end{bmatrix} \tag{3.97}$$

3.2.4. Compuertas SWAP y CSWAP

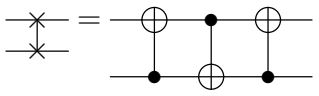
Otras compuertas importantes que se utilizaran a lo largo de este trabajo son las compuertas de intercambio SWAP y CSWAP. Esta última, también llamada compuerta de Fredkin, es la versión controlada de la anterior. En las figuras (3.98) y (3.99) respectivamente se muestran sus matrices y como se representan en circuitos cuánticos.

$$SWAP = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \tag{3.98}$$

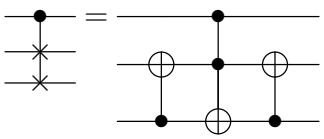
$$CSWAP = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} = \tag{3.99}$$



Ambas compuertas pueden implementarse usando compuertas CNOT y Toffoli y, por simplicidad, así se hará en este trabajo:



(3.100)



(3.101)

3.2.5. Compuertas de Medición

Retomando la discusión de la sección 3.1.5, para implementar una medición se tiene que primero calcular la probabilidad de medir cada uno de los posibles resultados, y luego, elegir uno de ellos

con esta probabilidad. La probabilidad de medir el i-ésimo qubit en el estado fundamental de un sistema de z qubits esta dada por la ecuación (3.42), la probabilidad de medir el estado excitado es $P(1) = 1 - P(0)$. Una vez se tienen estas probabilidades, se debe elegir de forma aleatoria uno de los dos estados, en base a estas probabilidades. Para esto, se sortea un número aleatorio x entre 0 y 1, y si dicho número satisface $x < P(0)$ se considera que se mide el qubit en el estado fundamental. Por otro lado, si $x > P(0)$ se considera el qubit en el estado excitado. Luego, se proyecta el vector de estado del sistema al autoestado de 0 o 1 según lo medido

$$|\psi'\rangle = \frac{P_0 |\psi\rangle}{\sqrt{P(0)}} \tag{3.102}$$

Para los operadores P_0 y P_1, se tiene que son simples operadores de un qubit

$$P_0 = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} = |0\rangle \langle 0| \tag{3.103}$$

$$P_1 = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} = |1\rangle \langle 1| \tag{3.104}$$

Como estos operadores no son unitarios, es necesario dividir por $\sqrt{P(0)}$ o $\sqrt{P(1)}$ según corresponda.

Para calcular la probabilidad $P(0)$ de medir el qubit i-ésimo en el estado fundamental, no es conveniente calcular directamente el elemento de matriz de $P(0) = \langle\psi| \left(I_{2^{i-1}} \otimes P_0 \otimes I_{2^{z-i}}\right) |\psi\rangle$, ya que de esta forma se realizarían muchas operaciones innecesarias. Como ya de antemano sabemos que coeficientes del vector de estado $|\psi\rangle$ corresponden al i-ésimo qubit en el estado fundamental, podemos sumar directamente solo sus módulos. Empecemos con el caso $i = 1$, entonces $P(0) = \langle\psi| \left(|0\rangle \langle 0| \otimes I_{2^{z-1}}\right) |\psi\rangle$. Sea $|\psi\rangle = \sum_{x=0}^{2^z-1} q(x) |x\rangle$, se tiene que

$$\begin{aligned} P(0) &= \sum_{x=0}^{2^z-1} q^*(x) \langle x| \left(|0\rangle \langle 0| \otimes I_{2^{z-1}}\right) \sum_{x'=0}^{2^z-1} q(x') |x'\rangle = \sum_{x,x'=0}^{2^z-1} q^*(x) q(x') \langle x| \left(|0\rangle \langle 0| \otimes I_{2^{z-1}}\right) |x'\rangle \\ &= \sum_{x,x'=0}^{2^z-1} q^*(x) q(x') \langle x| \left(I_{2^{z-1}} \oplus 0_{2^{z-1}}\right) |x'\rangle = \sum_{x,x'=0}^{2^{z-1}-1} q^*(x) q(x') \langle x| I_{2^{z-1}} |x'\rangle \\ &= \sum_{x,x'=0}^{2^{z-1}-1} q^*(x) q(x') \delta_{xx'} = \sum_{x=0}^{2^{z-1}-1} q^*(x) q(x') = \sum_{x=0}^{2^{z-1}-1} |q(x)|^2 \end{aligned} \tag{3.105}$$

Análogamente, si $i = 2$

$$\begin{aligned}
P(0) &= \sum_{x,x'=0}^{2^z-1} q^*(x)q(x') \langle x| \left(I_2 \otimes |0\rangle \langle 0| \otimes I_{2^{z-2}}\right) |x'\rangle = \sum_{x,x'=0}^{2^z-1} q^*(x)q(x') \langle x| \left(I_2 \otimes \left(I_{2^{z-2}} \oplus 0_{2^{z-1}}\right)\right) |x'\rangle \\
&= \sum_{j,j'=0}^{1} \sum_{x,x'=0}^{2^{z-1}-1} q^*(j2^{z-1}+x)q(j'2^{z-1}+x') \langle j2^{z-1}+x| \left(I_{2^{z-2}} \oplus 0_{2^{z-1}}\right) |j'2^{z-1}+x'\rangle \\
&= \sum_{j,j'=0}^{1} \sum_{x,x'=0}^{2^{z-2}-1} q^*(j2^{z-1}+x)q(j'2^{z-1}+x') \langle j2^{z-1}+x| I_{2^{z-2}} |j'2^{z-1}+x'\rangle \\
&= \sum_{j,j'=0}^{1} \sum_{x,x'=0}^{2^{z-2}-1} q^*(2j2^{z-2}+x)q(2j'2^{z-2}+x')\delta_{xx'}\delta_{jj'} = \sum_{j=0}^{1} \sum_{x=0}^{2^{z-2}-1} |q(2j2^{z-2}+x)|^2
\end{aligned} \tag{3.106}$$

Ya puede observarse un patrón. En general, para i arbitrario, se tiene que

$$P(0) = \sum_{j=0}^{a-1} \sum_{x=0}^{b-1} |q(2j2^{z-i}+x)|^2 \tag{3.107}$$

donde $a = 2^{i-1}$ y $b = 2^{z-i}$. Podría seguirse un desarrollo semejante para obtener $P(1)$, pero es mas sencillo usar que $P(1) = 1 - P(0)$.

Según el resultado de la medición sea 0 o 1, se procede a proyectar el vector de estado. Si se midió 0, se tiene que todos los coeficientes correspondientes al estado excitado tienen amplitud nula. Dichos coeficientes son $q((2j+1)b+x)$ con $j \in (0, 2^{i-1})$ y $x \in (0, 2^{z-i})$. Por otro lado, los coeficientes correspondientes al estado fundamental, es decir $q(2jb+x)$ con $j \in (0, 2^{i-1})$ y $x \in (0, 2^{z-i})$, ven su amplitud modificada en un factor $1/\sqrt{P(0)}$. Si se midió 1, en cambio, se anulan los coeficientes correspondientes al estado fundamental.

Capítulo 4

Algoritmos Cuánticos

En este capitulo se utilizaran los qubits, compuertas cuánticas y circuitos presentados en el capítulo anterior para recrear algoritmos clásicos en el modelo de computación cuántica y también, explorar las particularidades de la mecánica cuántica para llevar a cabo tareas computacionales de manera eficiente.

En la sección 4.1 se empieza hablando de como pasar de algoritmos clásicos a cuánticos y la necesidad de transformarlos en reversibles. Luego, se explora una característica fundamental de la computación cuántica, el *Paralelismo Cuántico*, y se expone su poder utilizándola en el algoritmo de Deutsch.
En la sección siguiente,4.2, se introduce la Transformada Cuántica de Fourier cuya superioridad en términos de eficiencia computacional con respecto a su análoga clásica le confiere una gran utilidad. Además, es necesaria en el procedimiento de *Estimación de Fase* que a su vez es fundamental para muchos algoritmos cuánticos, entre ellos el algoritmo de Shor.
En la sección 4.3 se construye el algoritmo cuántico más conocido, el algoritmo de Shor para la factorización de enteros grandes en sus factores primos. Allí se muestra como transformar este problema de factorización en un problema de encontrar el período de una función, la exponenciación modular. Se explica como aplicar la estimación de fase al problema particular de encontrar el periodo de la exponenciación modular, y se muestra el procesamiento posterior necesario para en base a los resultados de la estimación de fase conseguir los factores primos.
En las secciones 4.4 y 4.5 se presentan implementaciones completas del algoritmo de Shor, que incluyen algoritmos para la implementación eficiente de la exponenciación modular reversible a partir de compuertas fundamentales.

4.1. Introducción

¿Es posibles simular circuitos clásicos usando el modelo de los circuitos cuánticos? La respuesta, consistente con la intuición de que, en general, las leyes de la física clásica son un caso límite particular de las de la cuántica, es que si. Sin embargo, no puede hacerse directamente, debiendo primero superar el obstáculo de la reversibilidad. Como vimos en la sección 3.1.6, los circuitos cuánticos y clásicos difieren de manera fundamental en el requisito de reversibilidad de los primeros. En la sección 2.1 se puede ver que todas las compuertas lógicas fundamentales de dos bits son irreversibles, en tanto que solo tienen un bit de salida.

4.1.1. De Circuitos Clásicos a Cuánticos

Se demostrará que usando solo la compuerta de TOFFOLI o CCNOT es posible implementar las cuatro compuertas lógicas fundamentales de manera reversible, solo a costa de uno o dos bits de trabajo. La compuerta de Toffoli tiene tres bits, tanto de entrada como de salida, dos de los cuáles son *bits de control* que no son modificados, mientras que el tercero, *bit de acción* u *objetivo*, negado si y sólo si ambos bits de control son verdaderos o 1. En las figuras 4.2 y 4.1, se muestra la tabla de verdad y el circuito de la compuerta de Toffoli.

Entrada			Salida		
a	b	c	a'	b'	c'
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	1	0
0	1	1	0	1	1
1	0	0	1	0	0
1	0	1	1	0	1
1	1	0	1	1	1
1	1	1	1	1	0

Figura 4.1: Tablas de verdad

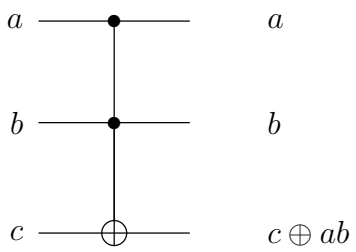


Figura 4.2: Circuito Cuántico

(4.1)

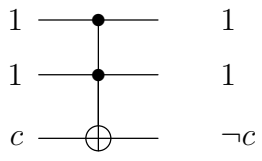
La compuerta de Toffoli cumple con la reversibilidad, ya que al aplicarla dos veces se regresa al estado inicial:

$$\begin{aligned} CCNOT(CCNOT(a,b,c)) &= CCNOT(a,b,c\oplus ab) = (a,b,(c\oplus ab)\oplus(c\oplus ab)) \\ &= (a,b,(c\oplus c)\oplus(ab\oplus c)\oplus(c\oplus ab)\oplus(ab\oplus ab) \\ &= (a,b,c\oplus(ab\oplus c)\oplus(ab\oplus c)) = (a,b,c) \end{aligned} \tag{4.2}$$

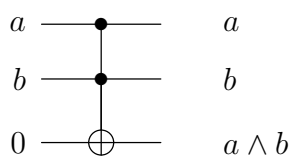
Luego, la compuerta de Toffoli es su propia inversa. Cabe notar que aqui se cumple que $ab = a \wedge b$ y $a \oplus b = a \veebar b$.

La compuerta de Toffoli se puede usar para implementar las cuatro compuertas básicas:

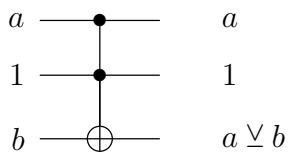
- *NOT*



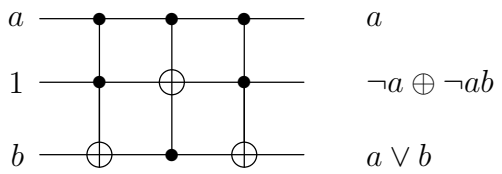
- *AND*



- *XOR*

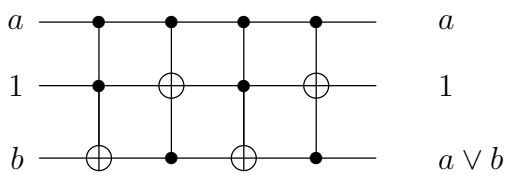


- *OR*



En el caso de *OR*, vemos que el segundo bit ya no tiene el estado inicial, si no que ahora tiene *basura* en el sentido de que es un estado no buscado. Esto es indeseable, ya que a pesar de estar usando compuertas de Toffoli que individualmente son reversibles, por simple inspección podemos ver que el circuito como un todo es irreversible ya que no se puede recorrer equivalentemente en ambos sentidos.

Para solucionar esto, hacemos uso de la reversibilidad de Toffoli, que implica que es su misma inversa, para deshacer el cambio que hemos hecho sobre el segundo bit al llevar a cabo la segunda compuerta. De esta forma, como se muestra a continuación, recuperamos el estado inicial del bit ancilla sin perder la operación lógica que queríamos obtener, la cuál se encuentra en el tercer qubit.



(4.3)

Esta técnica de volver a recorrer el camino de compuertas en sentido inverso una vez obtenida la operación deseada será usada intensamente a lo largo del trabajo.

4.1.2. Paralelismo Cuántico

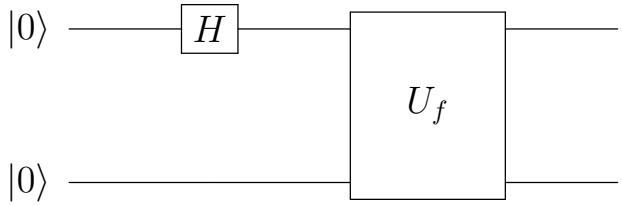
El *Paralelismo Cuántico* es una característica fundamental en muchos algoritmos cuánticos, y también de las más interesantes de la computación cuántica en general. Simplificando, el paralelismo cuántico permite evaluar una función $f(x)$ para distintos valores de x simultáneamente.

Supongamos $f(x) : \{0,1\} \to \{0,1\}$ una función booleana, podemos construir un circuito cuántico que la compute, utilizando dos qubits. Asumamos que existe una secuencia de compuertas capaz de realizar la transformación $|x,y\rangle \to |x, y \oplus f(x)\rangle$, o sea, un oráculo o caja negra U_f, asumiendo que U_f es unitaria y reversible.

Si el estado inicial es $|x\rangle = (|0\rangle + |1\rangle)/\sqrt{2} = H|0\rangle$, podemos ver que

$$\begin{aligned} &U_f |x,y\rangle = |x, y \oplus f(x)\rangle \Rightarrow \\ &U_f(|0,y\rangle + |1,y\rangle)/\sqrt{2} = (U_f|0,y\rangle + U_f|1,y\rangle)/\sqrt{2} = (|0, y\oplus f(0)\rangle + |1, y \oplus f(1)\rangle)/\sqrt{2} \end{aligned} \tag{4.4}$$

El sistema se encuentra en un superposición de estados $|x, y \oplus f(x)\rangle$ para $x = 0, 1$, o sea, hemos calculado $f(x)$ para ambos valores de x simultáneamente, pero esto habiendo ejecutado el circuito solo una vez. Solo fue necesario transformar el estado inicial en una superposición de estados, para luego ejecutar la compuerta U_f de la misma forma que si quisiéramos computar solo para un valor de x. A continuación se muestra el circuito cuántico descrito.



$$|\psi\rangle = \frac{|0, f(0)\rangle + |1, f(1)\rangle}{\sqrt{2}} \tag{4.5}$$

Notemos que como a la salida tenemos un estado entrelazado, no se puede explicitar cada qubit en su cable correspondiente dentro del circuito, si no que se debe mostrar el estado de todo el sistema.

Para poder explotar el paralelismo cuántico es necesario primero obtener un estado inicial entrelazado, para lo cuál se usa la compuerta de Hadamard. Esta acción puede extenderse a varios qubits, simplemente aplicando compuertas de Hadamard a cada uno de ellos en paralelo, por ejemplo:

$$H^{\otimes 2}|0,0\rangle = (H|0\rangle \otimes H|0\rangle) = \frac{|0,0\rangle + |0,1\rangle + |1,0\rangle + |1,1\rangle}{\sqrt{2}} \tag{4.6}$$

El caso de arriba puede generalizarse para cualquier función $f : \{0,1\}^n \to \{0,1\}^m$. Consideremos un sistema con dos *registros* $|x,y\rangle$. Aqui, $|x\rangle$ es el registro de entrada, esta compuesto por n qubits y $|y\rangle$ el de salida, compuesto por m qubits. Si se aplican compuertas de Hadamard a todos los

qubits del registro de entrada, estando estos originalmente en el estado fundamental $|0\rangle$, se obtiene una superposición simétrica de todos los 2^n estados posibles en el sentido de que todos los estados presentan la misma amplitud de probabilidad y la misma fase. Luego, podemos aplicar la caja negra U_f para calcular el valor de $f(x)$ para cada x. Si todos los qubits del registro de salida estaban *inicializados* a cero, es decir, inicialmente se encontraban en el estado $|0\rangle$, se tiene que el estado final es

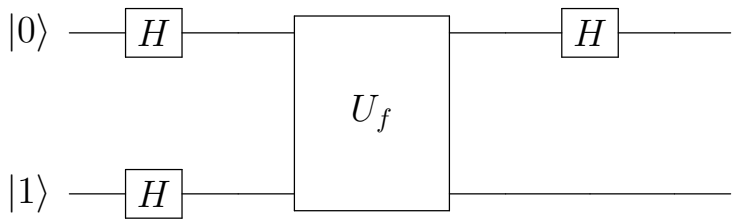
$$\frac{1}{\sqrt{2^n}} \sum_x |x, f(x)\rangle \tag{4.7}$$

Sin embargo, como vimos en la sección 3.1.2, no debemos olvidar que no toda la información contenida en un estado es útil, porque primero debemos medirlo. Volviendo al caso (4.4), vemos que si medimos el estado final obtenemos o bien $|0, f(0)\rangle$ o bien $|1, f(1)\rangle$, perdiéndose la información sobre el estado no medido. Aquí solo pudimos extraer de la información oculta el valor de $f(x)$ para un x en particular. Es muy importante en el diseño de los algoritmos cuántica las maximización de la información útil extraída, casi tanto como poder implementar una operación u otra. De todas formas, hay que repetir que esta información oculta sigue estando en el estado mientras este no sea medido, y aunque no podamos usarla directamente, puede ser útil de otras formas, por ejemplo para procesar información medible.

4.1.3. Algoritmo de Deutsch

Como primer ejemplo de un algoritmo con nombre por su relevancia histórica, vamos a ver el algoritmo de Deutsch. Basándonos en el circuito 4.1.2, construiremos un circuito que supera en desempeño a su mejor análogo clásico. El algoritmo de Deutsch combina el paralelismo cuántico con una propiedad de la mecánica cuántica heredada de la óptica llamada *interferencia*. Preliminarmente, introduzcamos la *base de Hadamard*, que surge de aplicar la compuerta de Hadamard a cada elemento de la base computacional: $|+\rangle = H|0\rangle = (|0\rangle + |1\rangle)/\sqrt{2}$ y $|-\rangle = H|1\rangle = (|0\rangle - |1\rangle)/\sqrt{2}$. Preparemos el estado inicial esta vez en el estado $|+,-\rangle$, aplicando compuertas de Hadamard paralelas sobre $|0,1\rangle$. Primero presentemos el circuito del algoritmo 4.8, y luego explicaremos paso a paso su ejecución.

$$|\psi_0\rangle \qquad |\psi_1\rangle \qquad |\psi_2\rangle \qquad |\psi_3\rangle \tag{4.8}$$



El estado inicial $|\psi_0\rangle = |01\rangle$ es transformado mediante compuertas de Hadamard

$$|\psi_1\rangle = |+\rangle \otimes |-\rangle \tag{4.9}$$

Un análisis por inspección nos permite notar que si aplicamos la compuerta U_f al estado $|x\rangle \otimes |-\rangle$ obtenemos el estado $(-1)^{f(x)} |x\rangle \otimes |-\rangle$. Luego,

$$|\psi_2\rangle = \begin{cases} \pm |+\rangle \otimes |-\rangle & \text{si } f(0) = f(1) \\ \pm |-\rangle \otimes |-\rangle & \text{si } f(0) \neq f(1) \end{cases} \tag{4.10}$$

Ejecutando la compuerta de Hadamard sobre el primer qubit vemos que

$$|\psi_3\rangle = \begin{cases} \pm |0\rangle \otimes |-\rangle & \text{si } f(0) = f(1) \\ \pm |1\rangle \otimes |-\rangle & \text{si } f(0) \neq f(1) \end{cases} \tag{4.11}$$

Como $f(0) \oplus f(1) = 1 \iff f(0) = f(1)$, podemos reescribir $|\psi_3\rangle$ como

$$|\psi_3\rangle = \pm |f(0) \oplus f(1)\rangle \otimes |-\rangle \tag{4.12}$$

Luego, midiendo el primer qubit podemos saber a través de $f(0) \oplus f(1)$ si $f(0) = f(1)$. Con solo una evaluación la función, el circuito nos ha permitido determinar una propiedad *global* de $f(x)$, cosa que requeriría al menos dos evaluaciones de $f(x)$ en un circuito clásico.

Este ejemplo nos permite aclarar las diferencias entre el paralelismo cuántico y los algoritmos clásicos probabilísticos. A priori uno podría asumir que evaluar el estado superpuesto $|0, f(0)\rangle + |1, f(1)\rangle$ se corresponde con un circuito clásico probabilístico que evalúa $f(x)$, con un probabilidad de que $x = 0$o1 de un medio. Sin embargo, estos algoritmos clásicos aún requieren de al menos dos evaluaciones, ya que una alternativa excluye a la otra, a diferencia del algoritmo cuántico en el que ambas alternativas *interfieren* una con la otra en una sola ejecución.

La esencia del poder de muchos algoritmos cuánticos descansa en una elección inteligente de las transformaciones finales que permitan determinar propiedades globales de la función estudiada, para las cuáles serían necesarias muchas evaluaciones en una computadora clásica.

El algoritmo de Deutsch es un caso particular del algoritmo de Deutsch-Joza, el cuál permite resolver el problema específico de determinar si una función $f : \{0,1\}^n \to \{0,1\}$ es constante o balanceada, es decir, si la mitad de los $x \in (0, 2^n - 1)$ dan como resultado 1 mientras que la otra mitad da 0 (solo estas dos posibilidades). El algoritmo de Deutsch-Joza permite resolver el problema con una sola evaluación de $f(x)$, en vez de $2^{n-1} + 1$ evaluaciones como en el peor de los casos con un algoritmo clásico.

Por mas interés práctico que tengan estos algoritmos, la verdad es que lo problemas que resuelven no son muy importantes, no teniendo hasta ahora ninguna utilidad práctica conocida. En las secciones siguientes exploraremos otros algoritmos cuánticos mucho mas útiles, los cuáles tienen algo en común, la *Transformada de Fourier Cuántica*.

4.2. Transformada de Fourier Cuántica

Una sección aparte merece la Transformada de Fourier Cuántica o QFT por sus siglas en inglés *Quantum Fourier Transform*, una operación muy importante debido a su utilidad matemática y extensamente utilizada tanto en computación cuántica como clásica.

Muchos problemas pueden resolverse mucho más fácil si se los *transforma* primero en otro tipo de problemas. Existen varias de estas transformaciones que permiten ir ida y vuelta de un tipo de problemas a otro, y constituyen herramientas tan poderosas que aparecen en muchos contextos distintos. Un descubrimiento clave en el campo de la computación Cuántica ha sido que algunas de estas transformaciones, como la transformada de Fourier, representan problemas más manejables desde el punto de vista de la complejidad computacional, proporcionando en algunos casos una *aceleración exponencial* (del inglés *exponential speed-up*, en un sentido que se explica mas adelante.

4.2.1. Transformada Discreta de Fourier

La QFT fue desarrollada a partir de la transformada Discreta de Fourier (DFT), que es una operación que representa una versión discreta de la transformada de Fourier. La transformada discreta de Fourier es una transformación que lleva un conjunto $x_0 \ldots, x_{N-1} \in \mathbb{C}$ en otro $y_0 \ldots, y_{N-1} \in \mathbb{C}$ de la siguiente manera

$$y_k \equiv \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} e^{(2\pi i j k)/N} x_j \tag{4.13}$$

Esta transformación tiene un enorme número de aplicaciones en muchas ramas de las ciencias, ya que permite descomponer una señal en sus componentes de frecuencia y viceversa, lo que permite el análisis y resolución de los mismos en el espacio de Fourier, o de frecuencia. El descubrimiento del algoritmo Transformada Rápida de Fourier o FFT por sus siglas en inglés permitió computar la transformada discreta de Fourier de una manera más eficiente al reducir su complejidad de $\mathcal{O}(N^2)$ a $\mathcal{O}(Nlog(N))$.

4.2.2. Definición

La QFT es la misma transformación que la DFT, pero implementada específicamente en el contexto teórico de la computación cuántica. En analogía con (4.13), la QFT puede ser implemen-

tada como una simple combinación de compuertas de Hadamard sobre un qubit y compuertas de rotación controladas con un solo qubit de control. En comparación con la transformada Discreta de Fourier clásica, que ante una entrada de tamaño $N = 2^n$ requiere del orden de $\mathcal{O}(n2^n)$ compuertas, la QFT solo requiere $\mathcal{O}(n^2)$ compuertas. Es la base de muchos algoritmos cuánticos y uno de los principales motivos detrás del poder de la computación Cuántica. [15]

La QFT actúa sobre un vector de estado de tamaño $N = 2^n$ $|\psi\rangle = \sum_{x=0}^{N-1} a_x |x\rangle$ y lo mapea al estado $|\varphi\rangle = \sum_{x=0}^{N-1} b_x |x\rangle$ donde los coeficientes b_x están dados por:

$$b_k = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} a_j \omega_N^{jk} \quad k = 0, \ldots, N-1 \quad \text{con} \quad \omega_N = e^{\frac{2\pi i}{N}} \tag{4.14}$$

Puede observarse que los coeficientes b_k son la transformada discreta de Fourier de las amplitudes a_x. Entonces, la QFT de un vector de la base computacional $|j\rangle$ con $j = 0, \ldots, N-1$ es, sea F_N el operador QFT:

$$F_N |j\rangle = \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} \omega_N^{jx} |x\rangle \tag{4.15}$$

En notación producto externo:

$$F_N = \frac{1}{\sqrt{N}} \sum_{i,j=0}^{N-1} \omega_N^{ij} |j\rangle \langle i| \tag{4.16}$$

y en notación matricial, la QFT puede definirse como una matriz unitaria F_N dada por:

$$F_N = \frac{1}{\sqrt{N}} \begin{bmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega & \omega^2 & \omega^3 & \cdots & \omega^{N-1} \\ 1 & \omega^2 & \omega^4 & \omega^6 & \cdots & \omega^{2(N-1)} \\ 1 & \omega^3 & \omega^6 & \omega^8 & \cdots & \omega^{3(N-1)} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \omega^{N-1} & \omega^{2(N-1)} & \omega^{3(N-1)} & \cdots & \omega^{(N-1)^2} \end{bmatrix} \tag{4.17}$$

Usando las siguientes representaciones binarias

$$\begin{aligned} j_1 j_2 \ldots j_m \equiv & j_1 2^{m-1} + j_2^{m-2} + \cdots + j_m = \sum_{k=1}^{m} 2^{m-k} j_k \\ 0.j_1 j_2 \ldots j_m \equiv & j_1/2 + j_2/4 + \cdots + j_m/2^m = \sum_{k=1}^{m} 2^{-k} j_k \end{aligned} \tag{4.18}$$

se sigue a partir de (4.15) que

$$\begin{aligned} F_N |j\rangle &= \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} \omega_N^{xj} |j\rangle = \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} e^{\frac{2\pi i j x}{N}} |j\rangle \\ &= \frac{1}{\sqrt{N}} \sum_{x_1=0}^{1} \cdots \sum_{x_{n-1}=0}^{1} e^{2\pi i j} (\sum_{l=1}^{n} x_l 2^{-l} |x_1 \dots x_n\rangle \\ &= \frac{1}{\sqrt{N}} \sum_{x_1=0}^{1} \cdots \sum_{x_{n-1}=0}^{1} \bigotimes_{l=1}^{n} e^{\frac{2\pi i j}{x}_l 2^{-l}} |x_l\rangle \\ &= \frac{1}{\sqrt{N}} \bigotimes_{l=1}^{n} \big(\sum_{x_l=0}^{1} e^{2\pi i j x_l 2^{-l}} |x_l\rangle \big) = \frac{1}{\sqrt{N}} \bigotimes_{l=1}^{n} \big(|0\rangle + e^{2\pi i j 2^{-l}} |1\rangle \big) \\ &= \frac{1}{\sqrt{N}} \Big[\big(|0\rangle + e^{2\pi i 0.j_n} |1\rangle \big) \otimes \big(|0\rangle + e^{2\pi i 0.j_{n-1} j_n} |1\rangle \big) \otimes \cdots \otimes \big(|0\rangle + e^{2\pi i 0.j_1 j_2 \dots j_n} |1\rangle \Big) \Big] \end{aligned} \tag{4.19}$$

Esta representación producto tensorial es útil a la hora de deducir un circuito eficiente, ya que se observa claramente como cada qubit es el resultado de una rotación sobre el vector $|+\rangle$. El valor de j determina de que forma particular se hará dicha rotación. Consideremos inicialmente el vector de estado $|j\rangle = |j_1 \dots j_n\rangle$ al cuál se le aplica la compuerta de Hadamard sobre el primer qubit. Entonces se obtiene

$$\frac{1}{2} \big(|0\rangle + e^{2\pi i 0.j_1} |1\rangle) \otimes |j_2 \dots j_n\rangle \tag{4.20}$$

Si ahora aplicamos sobre este primer qubit compuertas de rotación R_k controladas por el k-ésimo qubit, con $k = 2, \dots, n$, donde

$$R_k = \begin{bmatrix} 1 & 0 \\ 0 & e^{\frac{2\pi i}{2^k}} \end{bmatrix} \tag{4.21}$$

se llega al resultado de la notación producto para el último qubit:

$$\frac{1}{\sqrt{2}} \big(|0\rangle + e^{2\pi i 0.j_1 j_2 \dots j_n} |1\rangle) \otimes |j_2 \dots j_n\rangle \tag{4.22}$$

Continuando con el segundo qubit y procediendo de forma análoga, al aplicar la compuerta de Hadamard se obtiene:

$$\frac{1}{\sqrt{4}} \big((|0\rangle + e^{2\pi i 0.j_1 j_2 \dots j_n} |1\rangle) \otimes (|0\rangle + e^{2\pi i 0.j_2} |1\rangle) \otimes |j_3 \dots j_n\rangle \big) \tag{4.23}$$

Aplicando las rotaciones controladas, ahora con $k = 2, \dots, n$, se llega al resultado del $n-1$-ésimo qubit en la notación producto:

$$\frac{1}{\sqrt{4}} \big((|0\rangle + e^{2\pi i 0.j_1 j_2 \dots j_n} |1\rangle) \otimes (|0\rangle + e^{2\pi i 0.j_2 \dots j_n} |1\rangle) \otimes |j_3 \dots j_n\rangle \big) \tag{4.24}$$

Continuando con cada qubit, se observa que el último sólo necesita una compuerta de Hadamard y ninguna rotación para obtener el resultado correspondiente al primer qubit de la notación producto.

$$\frac{1}{\sqrt{N}}\left((|0\rangle + e^{2\pi i 0.j_1 j_2 \ldots j_n}|1\rangle) \otimes (|0\rangle + e^{2\pi i 0.j_2 \ldots j_n}|1\rangle) \otimes \ldots (|0\rangle + e^{2\pi i 0.j_n}|1\rangle\right) \tag{4.25}$$

Agregando entonces operaciones $SWAP$ entre los qubits i y $n-i$ con $i = 1, \ldots, n/2$ se obtiene la transformada Cuántica de Fourier del vector $|j\rangle$, como en la ecuación (4.19). Explicitando lo explicado usando circuitos, la QFT sobre n qubits se implementa de la siguiente manera:

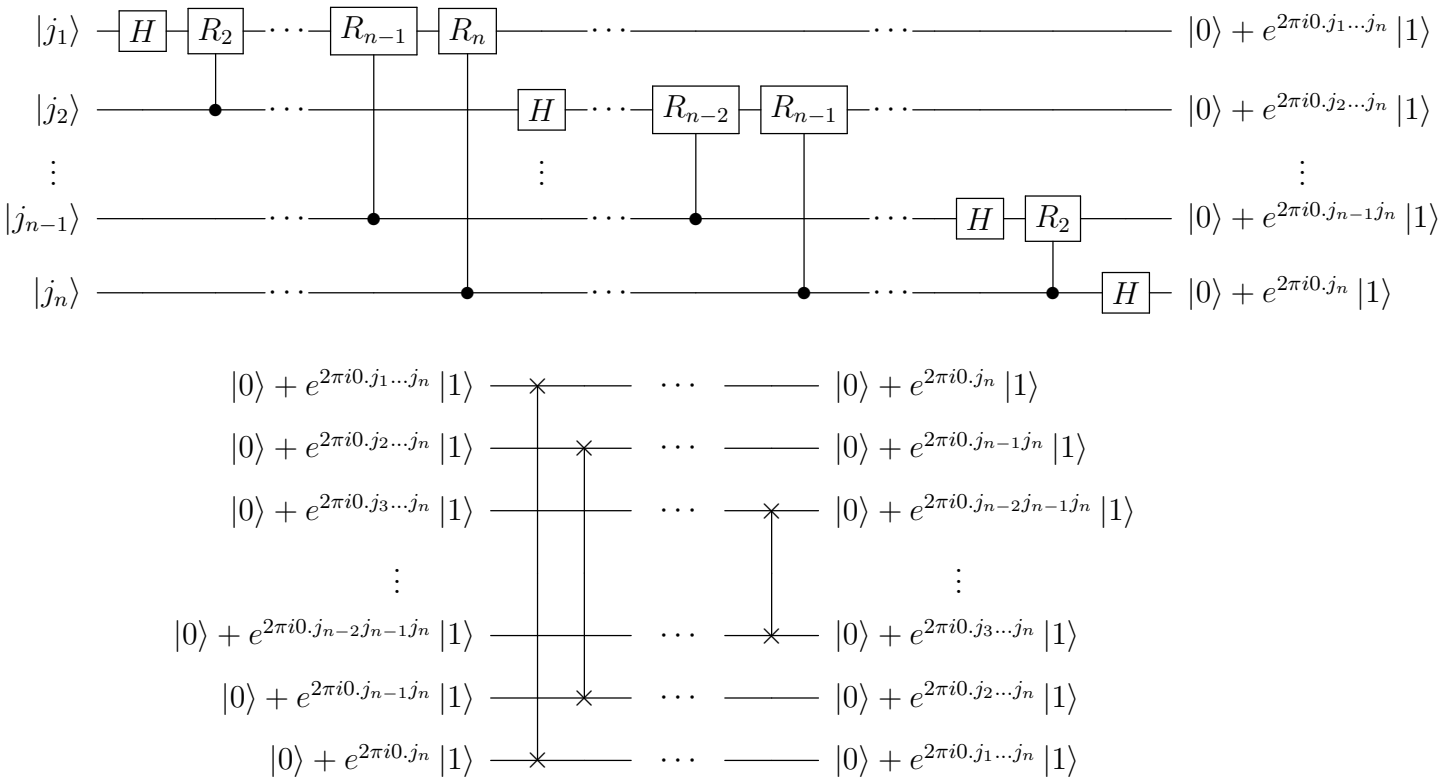


Figura 4.3: Circuito Cuántico que implementa la QFT. En el segundo circuito se observan las compuertas $SWAP$ que deben aplicarse para obtener el orden correcto de los qubits. Para mayor visibilidad no se muestran los factores $\frac{1}{\sqrt{N}}$ comunes a todos los qubits de salida

Como toda compuerta cuántica, la QFT es reversible. Por lo tanto existe una transformación unitaria inversa, la transformada Cuántica de Fourier inversa (QFT^{-1}) cuyo operador denotaremos como F_N^{-1}. Su expresión en notación producto puede deducirse como consecuencia del hecho de que $F_N F_N^{-1} = 1$ y, a partir de la ecuación (4.16):

$$F_N^{-1} = \frac{1}{\sqrt{N}} \sum_{i,j=0}^{N-1} \omega_N^{-ij} |j\rangle \langle i| \qquad \omega_N = e^{\frac{2\pi i}{N}} \tag{4.26}$$

y su matriz:

$$F_N^{-1} = \frac{1}{\sqrt{N}} \begin{bmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega^{-1} & \omega^{-2} & \omega^{-3} & \cdots & \omega^{-N-1} \\ 1 & \omega^{-2} & \omega^{-4} & \omega^{-6} & \cdots & \omega^{-2(N-1)} \\ 1 & \omega^{-3} & \omega^{-6} & \omega^{-8} & \cdots & \omega^{-3(N-1)} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \omega^{-(N-1)} & \omega^{-2(N-1)} & \omega^{-3(N-1)} & \cdots & \omega^{-(N-1)^2} \end{bmatrix} \tag{4.27}$$

La construcción del circuito cuántico para la QFT^{-1} es directa si se se tiene en cuenta que cada una de las compuertas con las que se implementa la QFT en la figura 4.3 tienen una inversa clara: $H^{-1} = H$, $R_k^{-1} = R_{-k}$ y $SWAP^{-1} = SWAP$. Entonces, teniendo en cuenta que las rotaciones controladas pueden intercambiarse los qubits de control y acción

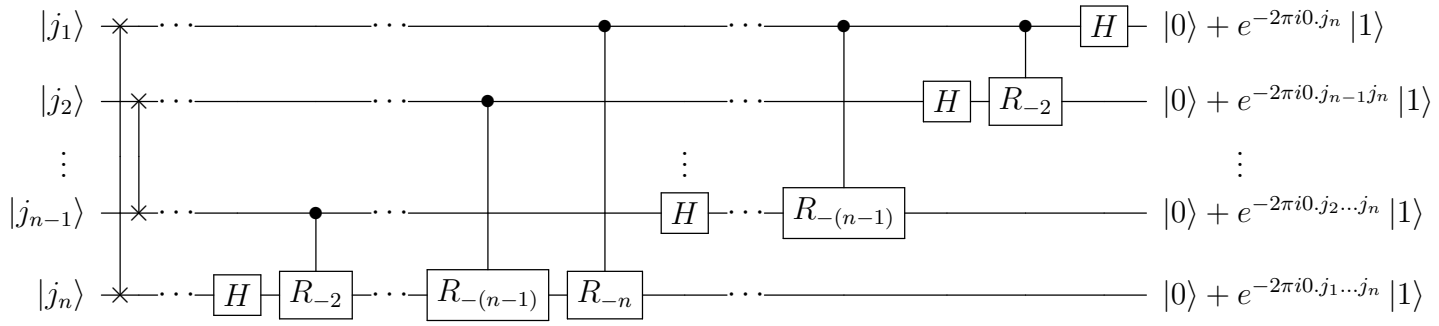


Figura 4.4: Circuito Cuántico que implementa la QFT^{-1}. No se muestran los factores $\frac{1}{\sqrt{N}}$ comunes a todos los qubits de salida

Como ya se dijo en la introducción del capítulo, uno de los aspectos fundamentales de la QFT es su escalabilidad comparada con la versión clásica. Revisando el circuito 4.3, podemos observar que sobre el qubit i-ésimo se le aplica una compuerta de Hadamard e $i-1$ compuertas de rotación. Sumando sobre todos los qubits, es decir $i = 1, \ldots, n$, se tiene que se requieren $n(n+1)/2$ compuertas. Además, a lo sumo se requieren $n/2$ compuertas $SWAP$ ($(n-1)/2$ si n es impar), cada una de las cuáles puede implementarse mediante 3 compuertas $CNOT$. Por lo tanto, vemos que este algoritmo escala como $\Theta(n^2)$. En comparación, el mejor algoritmo para calcular la transformada discreta de Fourier de 2^n elementos, la *Transformada Rápida de Fourier* (*FFT* por sus siglas en inglés), utiliza $\Theta(n2^n)$ compuertas. Es decir, este último requiere un número de compuertas exponencialmente mayor que la QFT.

4.2.3. Estimación de Fase

La QFT es la clave de un procedimiento llamado *estimación de fase* que es un elemento fundamental de una gran cantidad de algoritmos cuánticos. Sea U un operador unitario con autovalores y autovectores $e^{2\pi i\varphi}$ y $|u\rangle$ respectivamente. ¿Cómo puede obtenerse información sobre φ sin tener que repetir el algoritmo y medir muchas veces? Para estimar el valor de φ, supongamos que existen oráculos capaces de preparar el estado $|u\rangle$ y realizar la operación cU^{2^j}, es decir, la operación U^{2^j} controlada para cualquier entero j no negativo . El uso de cajas negras indican que el procedimiento no es un algoritmo completo a derecho propio, si no que depende de otros procedimiento o rutinas para poder ser implementado.

El procedimiento precisa de dos registros cuánticos, es decir, dos conjuntos de qubits. El primero consta de t qubits y se inicializa al estado $|0\rangle$, dependiendo t de la exactitud deseada en la estimación de fase y la probabilidad de medir el resultado deseado. El segundo registro se inicializa al estado $|u\rangle$ y posee los qubits necesarios para almacenar dicho estado, digamos n. En una primera etapa de la estimación de fases, se utiliza el hecho de que $U|u\rangle = e^{2\pi i\varphi}|u\rangle$ y se aplican al segundo registro los oráculos cU^{2^j} controlados por el qubit j-ésimo del primer registro, con $j = 1, \ldots, t$. Previa a la aplicación de los oráculos, se aplican compuertas de Hadamard sobre el primer registro. Es decir, para cierto j, se obtiene el estado

$$U^{2^j}|+, u\rangle = \frac{1}{\sqrt{2}}\big(|0\rangle + e^{2\pi i 2^j\varphi}|1\rangle\big) \otimes |u\rangle \tag{4.28}$$

La técnica utilizada es llamada coloquialmente *patear la fase* [17] y permite pasar de una fase global en el segundo qubit de la cuál es imposible obtener información alguna midiendo, a una fase relativa entre los estados $|0\rangle$ y $|1\rangle$ del qubit de control. Si se aplica una compuerta de Hadamard al primer qubit

$$\begin{aligned}\frac{1}{2}\big((1 + e^{2\pi i 2^j\varphi})|0\rangle + (1 - e^{2\pi i 2^j\varphi})|1\rangle\big) &= e^{\pi i 2^j\varphi}\Big(cos(2^j\pi\varphi)|0\rangle - i\ sin(2^j\pi\varphi)|1\rangle\Big) \\ &\equiv cos(2^j\pi\varphi)|0\rangle - i\ sin(2^j\pi\varphi)|1\rangle\end{aligned} \tag{4.29}$$

y luego se realiza un medición en este, la probabilidad de medir cada estado es

$$P(0) = |cos(2^j\pi\varphi)|^2 \quad P(1) = |sin(2^j\pi\varphi)|^2 \tag{4.30}$$

Entonces, la primera etapa de la estimación de fase consiste en la transformación

$$|0\rangle|u\rangle \implies \frac{1}{\sqrt{2^t}}\big(|0\rangle + e^{2\pi i 2^{t-1}\varphi}|1\rangle\big) \otimes \cdots \otimes \big(|0\rangle + e^{2\pi i 2^0\varphi}|1\rangle\big) \otimes |u\rangle = \frac{1}{\sqrt{2^t}}\sum_{k=0}^{2^t-1}(e^{2\pi i\varphi k}|k\rangle) \otimes |u\rangle \tag{4.31}$$

A fines ilustrativos, supongamos también que la fase φ puede escribirse exactamente con t cifras binarias como $\varphi = 0.\varphi_1 \dots \varphi_t$ según la ecuación (4.18), entonces

$$\begin{aligned} 2^j\varphi &= 2^j 0.\varphi_1 \dots \varphi_t \\ &= 2^j(\varphi_1/2 + \varphi_2/4 + \cdots + \varphi_t/2^t) \\ &= 2^{j-1}\varphi_1 + 2^{j-2}\varphi_2 + \cdots + 2^{j-t}\varphi_t \end{aligned} \tag{4.32}$$

Como $e^{2\pi i\varphi}e^{2^j\pi i\varphi} = e^{2^{j-1}\pi i\varphi}e^{2\pi i\varphi} \equiv e^{2\pi i\varphi}$ se ignoran los coeficientes 2^{j-k} tales que $k > j$ y se tiene que se recupera la representación decimal

$$e^{2\pi i 2^j\varphi} = e^{2\pi i 0.\varphi_{t-j}\dots\varphi_t} \tag{4.33}$$

Luego, la expresión (4.31) puede escribirse como

$$\frac{1}{\sqrt{2^t}}\big(\,|0\rangle + e^{2\pi i 0.\varphi_t}\,|1\rangle\,\big) \otimes \big(\,|0\rangle + e^{2\pi i 0.\varphi_{t-1}\varphi_t}\,|1\rangle\,\big) \otimes \cdots \otimes \big(\,|0\rangle + e^{2\pi i 0.\varphi_1\dots\varphi_t}\,|1\rangle\,\big) \otimes |u\rangle \tag{4.34}$$

Esta expresión nos es familiar. De (4.19), podemos ver que se trata del resultado de aplicar la QFT a estado $|\varphi\rangle = |\varphi_1 \dots \varphi_t\rangle$ de t qubits. Por lo tanto, es directo aplicar la inversa de la transformada de Fourier para entonces obtener dicho estado. Precisamente en esto último consiste la segunda etapa de la estimación de fase, aplicar la QFT^{-1} para obtener en el primer registro el estado $|\varphi_1 \dots \varphi_t\rangle$ que nos permite conocer exactamente la fase $\varphi = 0.\varphi_1 \dots \varphi_t$. Resumiendo, la acción del procedimiento de estimación de fase es realizar la transformación

$$|0\rangle \otimes |u\rangle \implies |\varphi\rangle \otimes |u\rangle \tag{4.35}$$

En la figura 4.5 se muestra un circuito que implementa el procedimiento de estimación de fase completo como se ha explicado

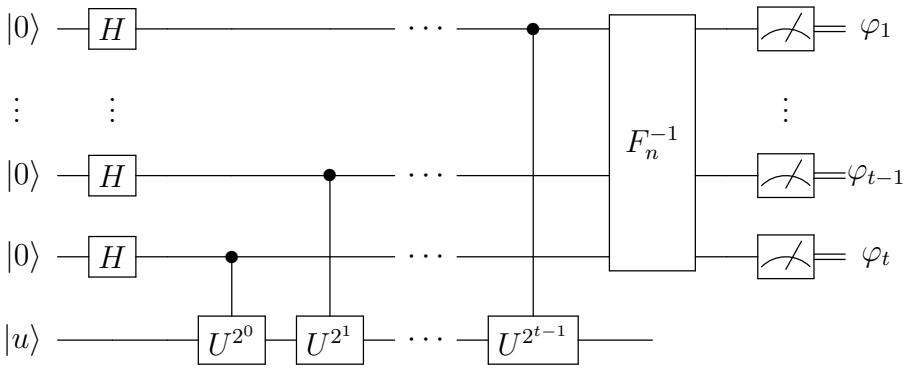


Figura 4.5: Circuito Cuántico que implementa el procedimiento de estimación de fase.

En el caso en que φ no pueda escribirse exactamente en t cifras binarias, el estado medido en el primer registro sera con probabilidad finita el mejor estimador de t bits de φ. Sea $c \in (0, 2^t - 1) \cap \mathbb{Z}$ el mejor estimador de t bits de φ, es decir, $\varphi = c/2^t + \delta$ donde $0 < \delta \leq 1/2^{t+1}$. Aplicando la QFT inversa a la ecuación (4.31), obtenemos en el primer registro

$$\begin{aligned}
&\frac{1}{2^t}\sum_{k=0}^{2^t-1}\sum_{y=0}^{2^t-1}\exp\Big(2\pi i y\Big(\frac{k}{2^t}-\varphi\Big)\Big)\,|k\rangle \\
&=\frac{1}{2^t}\sum_{k=0}^{2^t-1}\sum_{y=0}^{2^t-1}\exp\Big(2\pi i y\Big(\frac{k}{2^t}-\Big(\frac{c}{2^t}+\delta\Big)\Big)\Big)\,|k\rangle \\
&=\frac{1}{2^t}\sum_{k=0}^{2^t-1}\sum_{y=0}^{2^t-1}\exp\Big(\frac{2\pi i y(k-c)}{2^t}\Big)\exp(2\pi i y\delta)\,|k\rangle
\end{aligned} \tag{4.36}$$

El coeficiente que corresponde al estado $|c\rangle$ esta dado por la siguiente expresión, que es una suma geométrica.

$$\frac{1}{2^t}\sum_{y=0}^{2^t-1}(e^{2\pi i\delta})^y = \frac{1}{2^t}\left[\frac{1-(e^{2\pi i\delta})^{2^t}}{1-e^{2\pi i\delta}}\right] \tag{4.37}$$

Podemos corroborar del primer término que si $\delta = 0 \iff \varphi = c/2^t$, entonces la probabilidad de medir el estado es 1.
Como $\delta \leq 1/2^{t+1}$, vale que $2^{t+1}\pi\delta \leq \pi$, y por lo tanto podemos acotar el numerador como

$$|1 - e^{2\pi i\delta 2^t}| \geq \frac{2^{t+1}\pi\delta}{\pi/2} = 4\delta 2^t \tag{4.38}$$

También, $|1 - e^{2\pi i\delta}| \leq 2\pi\delta$, y por lo tanto la probabilidad de medir $|c\rangle$ es

$$\left|\frac{1}{2^t}\left[\frac{1-(e^{2\pi i\delta})^{2^t}}{1-e^{2\pi i\delta}}\right]\right| \geq \left[\frac{1}{2^t}\frac{4\delta 2^t}{2\pi\delta}\right]^2 = \frac{4}{\pi^2} \sim 0{,}405 \tag{4.39}$$

La probabilidad deducida arriba es la de medir un estimado exacto a t bits, habiendo usado t qubits en el primer registro. Si en cambio tomamos $m < t$, se tiene que la probabilidad de medir con exactitud ahora m bits de la fase φ aumenta. Sea l tal que $-2^{t-1} \leq l < 2^{t-1}$, denotemos α_l la amplitud del estado $|(c - l) \bmod 2^t\rangle$. Entonces, retomando la ecuación (4.37)

$$\alpha_l = \frac{1}{2^t}\left[\frac{1-(e^{2\pi i(\delta+l/2^t)})^{2^t}}{1-e^{2\pi i(\delta+l/2^t)}}\right] \tag{4.40}$$

Como

$$|1 - e^{2\pi i(\delta+l/2^t)}| \geq \frac{2\pi(\delta + l/2^t)}{pi/2} = 4(\delta + l/2^t) \tag{4.41}$$

entonces

$$|\alpha_l| \leq \left| \frac{2}{2^t 4(\delta + l/2^t)} \right| \leq \frac{2}{2^t 4(\delta + l/2^t)} = \frac{1}{2(2^t\delta + l)} \tag{4.42}$$

La probabilidad de obtener un error mayor que $k/2^t$, es decir $|c - x| > k$, es la suma de las probabilidades de todos los estados tales que $|l - x| > k$, o sea

$$\begin{aligned} P(|c-x| > k) &= \sum_{k \leq l < 2^{t-1}} |\alpha_l|^2 + \sum_{-2^{t-1} \leq l < k} |\alpha_l|^2 \\ &\leq \sum_{l=k}^{2^{t-1}-1} \frac{1}{4(l+2^t\delta)^2} + \sum_{l=-2^{t-1}}^{-(k+1)} \frac{1}{4(l+2^t\delta)^2} \\ &\leq \sum_{l=k}^{2^{t-1}-1} \frac{1}{4l^2} + \sum_{l=k+1}^{2^{t-1}} \frac{1}{4(l-1/2)^2} \\ &\leq \sum_{l=2k}^{2^{t-1}-1} \frac{1}{4(l/2)^2} \\ &< \int_{2k-1}^{2^t-1} \frac{1}{l^2} \\ &< \frac{1}{2k-1} \end{aligned} \tag{4.43}$$

donde en la tercera linea usamos que $0 < \delta \leq 1/2^{t+1}$. Esto implica que la probabilidad de medir x tal que difiera menos que k del mejor estimador c es

$$P(|x - c| < k) \leq 1 - \frac{1}{2k-1} = 1 - \epsilon \tag{4.44}$$

Sea $k = 2^p$, entonces vemos que $|x - c| < k \implies |x - c| < 2^p$. Puede verse más fácilmente en el caso particular de que $p = 0 \implies k = 1$ que esto significa que x es exacto a c hasta $m = t - p$ bits. Luego, la probabilidad de medir con exactitud $m = t - p$ bits es

$$P(|x - c| < 2^{t-m}) \leq 1 - \frac{1}{2^{t-m+1} - 1} = 1 - \epsilon \tag{4.45}$$

Si en cambio, quiero medir m bits exactos con cierta probabilidad finita $1 - \epsilon$, entonces el número de qubits del primer registro deben ser

$$\begin{aligned} \epsilon &= \frac{1}{2^{t-m+1} - 1} \\ 2^{t-m+1} &= 1/\epsilon + 1 \\ t - m + 1 &= \log(1 + 1/\epsilon) \\ t &= m - \log 2 + \log(1 + 1/\epsilon) \\ \implies t &= m + \left\lceil \log\Big(\frac{1}{2} + \frac{1}{2\epsilon}\Big) \right\rceil \end{aligned} \tag{4.46}$$

Vale la pena destacar que como el segundo registro debe ser inicializado al autoestado $|u\rangle$, es necesario conocer los autoestados del operador U y ser capaz de preparar el segundo registro en este estado. Esta puede ser una condición demasiado fuerte, pero como se verá a continuación, puede ser sorteada al costo de algo de aleatoriedad en el algoritmo. Supongamos inicializamos el segundo registro al vector $|\psi\rangle = \sum_i a_i |u_i\rangle$, donde $|u_i\rangle$ son los autoestados de U, cada uno con autovalores $e^{2\pi i\varphi_i}$, con a_i desconocidos. Por la linealidad de los operadores, es trivial ver a partir de la ecuación (4.35) que el procedimiento realiza la transformación

$$|0\rangle \otimes \sum_i a_i |u_i\rangle \implies \sum_i \left(a_i |\tilde{\varphi_i}\rangle \otimes |u_i\rangle \right) \tag{4.47}$$

donde $\tilde{\varphi_i} = \tilde{\varphi_{i1}} \dots \tilde{\varphi_{it}}$ es una aproximación de la fase φ_i exacta hasta t bits.

4.3. Algoritmo de Shor

Todos hemos escuchado alguna vez frases como *'la computación cuántica va a destruir la seguridad en internet'* o similares en la prensa sensacionalista. Aunque estas sean exageraciones para llamar la atención, *click-baits*, la verdad es que no están del todo erradas. El algoritmo de Shor es potencialmente capaz de dejar obsoleto el sistema criptográfico RSA, el algoritmo de cifrado de clave pública más utilizado en la actualidad. En el apéndice A se da una breve introducción al sistema RSA y se explica como la complejidad del problema de factorización de números enteros constituye la base de la fortaleza del mismo.

4.3.1. Aplicaciones de la QFT y la Estimación de Fase

La estimación de fases puede ser usada para resolver una gran variedad de interesantes problemas. Nos centraremos en el problema de encontrar el orden, del cuál se desprende la aplicación mas interesante: la factorización.

Se sabe desde muy antiguo que todo número puede descomponerse en un producto de primos. Incluso estos, que por definición solo son divisibles por 1 y por si mismos. Los números primos son intrínsecamente interesantes para los matemáticos, ya que hasta el día de hoy no los entendemos completamente en cuestiones como por ejemplo su aparente aleatoria distribución. Desde siempre nos hemos interesado en el problema de como descomponer o factorizar un número en sus primos. No fue hasta la segunda mitad del siglo XIX que fue analizado desde el punto de vista formal de las ciencias de la computación, lo que llevo a un gran avance en los algoritmos para factorizar. El mejor algoritmo hasta la fecha para factorizar enteros grandes (más de 100 cifras) es la *criba del cuerpo de números*. Heurísticamente[1], el mismo tiene a la hora de factorizar un número N de n

[1]Heurísticamente se refiere a una aproximación más práctica que óptima a la hora de abordar el problema.

bits, osea $2^{n-1} \leq N < 2^n$, una complejidad dada por $\mathcal{O}(\exp(n^{1/3}(\log n)^{2/3}))$, por lo que se dice que es exponencial en el tamaño del número.

A continuación describiremos el algoritmo, presentado por primera por Peter W. Shor [1], para la factorización de enteros grandes en sus factores primos. El mismo se basa en un subrutina para obtener el período de una función que conlleva $O(n^2(\log n)(\text{loglog } n))$ pasos en un computadora cuántica, más un procesamiento posterior polinomial en n en una computadora clásica.

4.3.2. Factorización y el Problema de hallar el Orden

La subrutina cuántica se encargar precisamente de encontrar el orden de un elemento a en el grupo multiplicativo mod N; es decir, el menor entero r tal que $a^r \equiv 1 \bmod N$. Esto es equivalente a encontrar el período de la función $f(x) = a^x \bmod N$, ya que $f(x+r) = a^{x+r} \bmod N = a^x \bmod N = f(x)$. Se demostrará a continuación que el problema de factorización puede ser reducido a este problema de hallar el orden de un elemento.

Para encontrar un factor no trivial de un entero impar N, suponiendo existe un oráculo para computar el orden r de cierto a, se debe obtener dicho orden y luego calcular el máximo divisor común $\gcd(a^{r/2} \pm 1, N)$, es decir, el máximo entero que es divide tanto a $a^{r/2} \pm 1$ como a N. Podemos usar el algoritmo de Euclides para computar el máximo divisor común en tiempo polinómico. Como $(a^{r/2}-1)(a^{r/2}+1) = a^r - 1 \equiv 0 \bmod N$, se tiene que $\gcd(a^{r/2} \pm 1, N)$ es un divisor no trivial de N excepto si r es impar o si $a^{r/2} \equiv -1 \bmod N$. Con esto en mente, el procedimiento aplicado a un $a \in [2, N-1]$ al azar tal que sea coprimo con N, da como resultado un factor primo no trivial de N A continuación se resume y esquematiza el algoritmo de factorización para un N general.

Algorithm 3 Algoritmo de factorización de Shor

Entrada: Un número compuesto N

Salida: Un factor no trivial de N

Ejecución:

1. Si N es par, devuelve el factor 2.
2. Determinar clásicamente si $N = c^b$ con enteros $c \geq 1$ y $b \geq 2$ y de ser así devolver el factor a.
3. Sortear un entero $a \in [2, N-1]$. Si el múltiplo común menor computado clásicamente cumple $\gcd(a, N) > 1$, devolver el factor $\gcd(a, N)$.
4. Usar la subrutina cuántica para hallar el orden r de $a \bmod N$.
5. Si r es par y $a^{r/2} \neq -1 \bmod N$ entonces computar $a^{r/2} \pm 1 \bmod N$ y comprobar si se trata de un factor no trivial. En caso contrario, repetir a partir de 3.

Los pasos 1 y 3 aseguran que N es un entero impar con mas de un factor primo, en caso de no dar un factor, y son computables clásicamente en tiempo polinómico.

Se considera que el problema de hallar el orden como un problema difícil para la computación clásica, ya que no se conocen algoritmos capaces de resolverlo usando recursos que escalen polinómicamente en n, además de que aplicado al caso de la factorización es menos eficiente que la criba del campo de números. Usando la estimación de fase, es posible obtener un algoritmo cuántico eficiente para el problema de hallar el orden.

El algoritmo cuántico para resolver el problema de hallar el orden es básicamente aplicar el procedimiento de estimación de fase aplicado al operador unitario

$$U_a |x\rangle = |ax \bmod N\rangle \qquad x = 0, \ldots, N-1 \tag{4.48}$$

Este operador tiene autovalores $e^{2\pi ij/r}$ y autovectores

$$|\psi_j\rangle = \frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} e^{-2\pi ijk/r} |a^k \bmod N\rangle \tag{4.49}$$

donde r cumple $a^r \bmod N = 1$, ya que

$$\begin{aligned} U_a |\psi_j\rangle &= \frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} e^{-2\pi ijk/r} |a^{k+1} \bmod N\rangle \\ &= \frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} e^{-2\pi ij(k-1)/r} |a^k \bmod N\rangle = e^{2\pi ij/r} |\psi_j\rangle \end{aligned} \tag{4.50}$$

Entonces, replicando el procedimiento de estimación de fase de la subsección 4.2.3 tenemos que el efecto de aplicar el operador controlado cU_a sobre el estado $|+, \psi_j\rangle$ es

$$cU_a |+, \psi_j\rangle = \frac{|0\rangle + e^{2\pi i\varphi} |1\rangle}{\sqrt{2}} \otimes |\psi_j\rangle \tag{4.51}$$

pateando efectivamente la fase al primer qubit.

Un problema al que nos enfrentamos es que no se pueden construir los autoestados y autovalores de U_a sin previo conocimiento de r, pero esto puede salvarse haciendo uso de la siguiente propiedad. Se puede demostrar que

$$\begin{aligned} \sum_{j=0}^{r-1} |\psi_k\rangle &= \frac{1}{r} \sum_{j=0}^{r-1} \sum_{k=0}^{r-1} e^{-2\pi ikj/r} |a^k \bmod N\rangle \\ &= \frac{1}{r} \sum_{k=0}^{r-1} \sum_{j=0}^{r-1} (e^{-2\pi ik/r})^j |a^k \bmod N\rangle \end{aligned} \tag{4.52}$$

Usando que $\sum_{j=0}^{r-1}(e^{-2\pi ik/r})^j = r\delta_{k,0}$ obtenemos que

$$\begin{aligned} &= \frac{1}{r}\sum_{k=0}^{r-1} r\delta_{k,0}\,|a^k \bmod N\rangle \\ &= |a^0 \bmod N\rangle = |1\rangle \end{aligned} \tag{4.53}$$

y luego, aplicando cU_a al estado $|1\rangle$ se obtiene el estado entrelazado

$$cU_a\,|+,1\rangle = \frac{1}{\sqrt{2}}\sum_{j=0}^{r-1}\Big((|0\rangle + e^{2\pi ij/r}\,|1\rangle)\Big)\otimes|\psi_j\rangle \tag{4.54}$$

Entonces, al aplicar el procedimiento de estimación de fase con este operador U_a, es necesario t qubits inicializados a cero en el primer registro y n qubits en el segundo registro, preparados en el estado de n qubits $|1\rangle = |0\ldots1\rangle$. Luego, se entrelazan los estados del primer registro usando compuertas de Hadamard y se aplican sucesivamente las compuertas controladas $cU_a^{2^j}$ con $j = 0,\ldots,t-1$, obteniéndose el estado

$$\frac{1}{2^t}\sum_{j=0}^{r-1}\big(\,|0\rangle + e^{2\pi 2^{t-1}ij/r}\,|1\rangle\,\big)\otimes\sum_{j=0}^{r-1}\big(\,|0\rangle + e^{2\pi 2^{t-2}ij/r}\,|1\rangle\,\big)\otimes\cdots\otimes\sum_{j=0}^{r-1}\big(\,|0\rangle + e^{2\pi 2^{0}ij/r}\,|1\rangle\,\big)\otimes|\psi\rangle \tag{4.55}$$

Como en la ecuación (4.34), esto puede reescribirse como

$$\frac{1}{2^t}\sum_{j=0}^{r-1}\big(\,|0\rangle + e^{2\pi i0.\varphi_{jt}}\,|1\rangle\,\big)\otimes\sum_{j=0}^{r-1}\big(\,|0\rangle + e^{2\pi i0.\varphi_{jt-1}\varphi_{jt}}\,|1\rangle\,\big)\otimes\cdots\otimes\sum_{j=0}^{r-1}\big(\,|0\rangle + e^{2\pi i0.\varphi_{j1}\ldots\varphi_{jt}}\,|1\rangle\,\big)\otimes|\psi\rangle \tag{4.56}$$

donde $j/r \sim \varphi_j = 0.\varphi_{j1}\ldots\varphi_{jt}$. De esta forma, según vimos en la subsección 4.2.3, hemos codificado en la fase del primer registro una superposición de estimados $\varphi_j \sim j/r$ con $j = 0,\ldots,r-1$.

El siguiente paso es aplicar la transformada inversa de Fourier para desentrelazar el primer registro. Se obtiene una superposición

$$\sum_{j=0}^{r-1}|\varphi_j\rangle\ \otimes|\psi\rangle \tag{4.57}$$

donde luego al medir obtenemos un estimador $c/2^l \sim j/r$ exacto hasta m bits con una probabilidad de al menos $1/r - 1/(r2^{t-m+1}-r)$ según la ecuación (4.45). En el circuito 4.6 se muestra el circuito del procedimiento de estimación de fase aplicado al algoritmo de Shor.

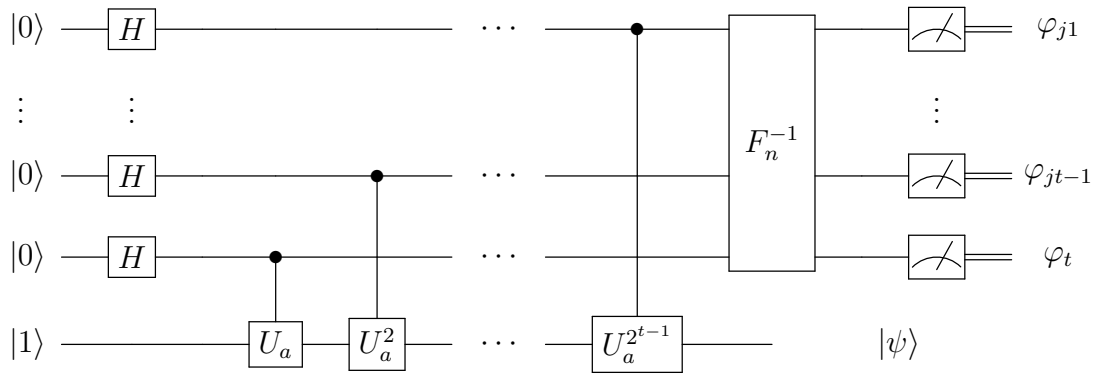


Figura 4.6: Circuito Cuántico del algoritmo de Shor.

La probabilidad de medir cada valor del estimador c se deduce de la siguiente forma. Luego de ejecutar las compuertas $cU_{a^{2^j}}$, el estado del sistema es el siguiente

$$|\psi\rangle = \frac{1}{\sqrt{q}} \sum_{x=0}^{q-1} |x\rangle \, |a^x \bmod N\rangle \tag{4.58}$$

donde $q = 2^t$. Como $N < 2^n$, entonces los estados $a^x \bmod N < N < 2^n$. Luego, si el periodo de $a^x \bmod N$ es r, entonces $\exists\ M \sim q/r$ valores de $0 \leq x < q$ tales que $a^x \bmod N = k$. Entonces, si el menor x que cumple esto es x_{0k}, los demás se forman como $x_{0k} + dr$, con $d = 0, \ldots, M-1$. También, sólo existen r valores de k, $a^i \bmod N$ con $i = 0, \ldots, r-1$, ya que tenemos que $Mr = q$,

$$\frac{1}{\sqrt{q}} \sum_{x=0}^{q-1} |x\rangle \, |a^x \bmod N\rangle = \frac{1}{\sqrt{q}} \sum_{i=0}^{r-1} \left(\sum_{d=0}^{M-1} |x_{0i} + dr\rangle \right) |a^i \bmod N\rangle \tag{4.59}$$

Cuando aplicamos la transformada inversa de Fourier al primer registro, obtenemos

$$\begin{aligned} |\psi\rangle &= \frac{1}{q} \sum_{i=0}^{r-1} \left(\sum_{c=0}^{q-1} \sum_{d=0}^{M-1} e^{(2\pi i c(x_{0i}+dr)/q)} |c\rangle \right) |a^i \bmod N\rangle \\ &= \frac{1}{q} \sum_{i=0}^{r-1} \left(\sum_{c=0}^{q-1} e^{(2\pi i c x_{0i}/q)} \left(\sum_{d=0}^{M-1} e^{(2\pi i\ cdr/q)} \right) |c\rangle \right) |a^i \bmod N\rangle \\ &= \frac{1}{q} \sum_{i=0}^{r-1} \left(\sum_{c=0}^{q-1} e^{(2\pi i c x_{0i}/q)} \left(\sum_{d=0}^{M-1} \zeta^d \right) |c\rangle \right) |a^i \bmod N\rangle \end{aligned} \tag{4.60}$$

donde $\zeta = e^{(2\pi i\ cr/q)}$. Para hallar el coeficiente de $|c\rangle$

$$
\begin{aligned}
&= \frac{1}{q}\sum_{c=0}^{q-1}\sum_{i=0}^{r-1} e^{2\pi ic}e^{(x_{0i}/q)}\left(\sum_{d=0}^{M-1}\zeta^d\right)|c\rangle\,|a^i \bmod N\rangle \\
&= \frac{1}{\sqrt{qMr}}\sum_{c=0}^{q-1} e^{2\pi ic}\left(\sum_{d=0}^{M-1}\zeta^d\right)|c\rangle\left(\sum_{i=0}^{r-1} e^{(x_{0i}/q)}\,|a^i \bmod N\rangle\right) \\
&= \frac{1}{\sqrt{qM}}\sum_{c=0}^{q-1} e^{2\pi ic}\left(\sum_{d=0}^{M-1}\zeta^d\right)|c\rangle\,\frac{1}{\sqrt{r}}\left(\sum_{i=0}^{r-1} e^{(x_{0i}/q)}\,|a^i \bmod N\rangle\right)
\end{aligned}
\tag{4.61}
$$

Luego, la probabilidad de medir cada valor de $c = 0, \ldots, 2^t - 1$ es

$$
P(c) = \left|\frac{1}{\sqrt{qM}}e^{2\pi ic}\left(\sum_{d=0}^{M-1}\zeta^d\right)\right|^2 = \frac{1}{qM}\left|\sum_{d=0}^{M-1}\zeta^d\right|^2 \tag{4.62}
$$

Los términos $\zeta = e^{(2\pi i\ cr/q)^d}$ son casi nulos excepto si c cumple que $c/q \sim j/r$, ya que $2\pi i\ cr/q \sim 2j\pi i \ \equiv 0 \implies \zeta = 1$. El estado final del registro de salida $|\psi\rangle$, que no es relevante pues la información sobre r esta codificada en la fase del primer registro, es como vimos una superposición de estados $|a^j \bmod N\rangle$ con $j = 0, \ldots, 2^t - 1$.

4.3.3. Exponenciación Modular Reversible

En la subsección anterior hemos explicado como aplicar el procedimiento de estimación de fase en el caso particular de U_a, pero no hemos explicitado como computar dicho operador. Veremos que esto resulta un cuello de botella en términos de complejidad computacional para el algoritmo de Shor. La secuencia de operadores $cU_a^{2^0}, cU_a^{2^1}, \ldots, cU_a^{2^{t-1}}$ implementa la operación de exponenciación modular, ya que cada $cU_a^{2^j}$ multiplica transforma el estado del segundo registro $|y\rangle \to |ya^{2^j} \bmod N\rangle$. Es decir, sean x_j con $j = 0, \ldots, t-1$ las cifras binarias de x, las compuertas controladas $cU_a^{2^j}$ solo se ejecutan si $x_j = 1$

$$
\begin{aligned}
|x\rangle \otimes |y\rangle &\to |x\rangle \otimes cU_a^{2^{t-1}} \ldots\ cU_a^{2^0}\,|y\rangle \\
&= |x\rangle \otimes |a^{x_{t-1}2^{t-1}} \ldots\ a^{x_0 2^0} y \bmod N\rangle \\
&= |x\rangle \otimes |a^{x_{t-1}2^{t-1}+\cdots+x_0 2^0} y \bmod N\rangle \\
&= |x\rangle \otimes |a^x y \bmod N\rangle
\end{aligned}
\tag{4.63}
$$

por lo que si el estado inicial del segundo registro es $|y\rangle = |1\rangle$, se obtiene $|a^x \bmod N\rangle$.
Esta operación debe ser llevada a cabo de forma reversible, mediante circuitos basados en los operadores básicos de la sección 3.2. Básicamente, se implementa cada $cU_a^{2^j}$ utilizando un tercer registro para calcular reversiblemente $a^{x_j 2^j} \bmod N$ y luego se multiplica reversiblemente el contenido del

segundo registro por esa cantidad. Finalmente, como se hizo en la sección 4.1.1, se borra el tercer registro aplicando la secuencia inversa de operaciones usadas para calcular $a^{2^j} \bmod N$.

$$\begin{aligned} cU_a^{2^j}(|y\rangle \otimes |0\rangle) &\rightarrow |y\rangle \otimes |a^{x_j 2^j} \bmod N\rangle \\ &\rightarrow |ya^{x_j 2^j} \bmod N\rangle \otimes |a^{x_j 2^j} \bmod N\rangle \\ &\rightarrow |ya^{x_j 2^j} \bmod N\rangle \otimes |0\rangle \end{aligned} \tag{4.64}$$

Entonces, las transformaciones sucesivas sobre los ahora tres registros que se llevan a cabo para completar la exponenciación modular son

$$\begin{aligned} |x\rangle \otimes |1\rangle \otimes |0\rangle &\rightarrow |x\rangle \otimes |1\rangle \otimes |a^{x_0 2^0} \bmod N\rangle \\ &\rightarrow |x\rangle \otimes |1a^{x_0 2^0} \bmod N\rangle \otimes |a^{x_0 2^0} \bmod N\rangle \\ &\rightarrow |x\rangle \otimes |1a^{x_0 2^0} \bmod N\rangle \otimes |0\rangle \\ &\rightarrow |x\rangle \otimes |a^{x_0 2^0} \bmod N\rangle \otimes |a^{x_1 2^1} \bmod N\rangle \\ &\rightarrow |x\rangle \otimes |(a^{x_0 2^0} a^{x_1 2^1}) \bmod N\rangle \otimes |a^{x_1 2^1} \bmod N\rangle \\ &\rightarrow |x\rangle \otimes |a^{x_1 2^1 + x_0 2^0} \bmod N\rangle \otimes |0\rangle \\ &\qquad\qquad \vdots \\ &\rightarrow |x\rangle \otimes |a^{x_{t-2} 2^{t-2} + \cdots + x_0 2^0} \bmod N\rangle \otimes |0\rangle \\ &\rightarrow |x\rangle \otimes |a^{x_{t-2} 2^{t-2} + \cdots + x_0 2^0} \bmod N\rangle \otimes |a^{x_{t-1} 2^{t-1}} \bmod N\rangle \\ &\rightarrow |x\rangle \otimes |a^{x_{t-1} 2^{t-1} + \cdots + x_0 2^0} \bmod N\rangle \otimes |a^{x_{t-1} 2^{t-1}} \bmod N\rangle \\ &\rightarrow |x\rangle \otimes |a^{x_{t-1} 2^{t-1} + \cdots + x_0 2^0} \bmod N\rangle \otimes |0\rangle \\ &\rightarrow |x\rangle \otimes |a^x \bmod N\rangle \otimes |0\rangle \end{aligned} \tag{4.65}$$

Como vemos, lograr la exponenciación modular de formar reversible requiere aumentar el número de qubits con al menos n qubits ancilla al ser necesario un tercer registro. Es más, todavía no se ha deducido como implementar las operaciones del cómputo de $a^{x_j 2^j} \bmod N$ y la multiplicación de esto por el segundo registro de manera reversible y a partir de compuertas fundamentales. Es por esto que se dice que la exponenciación modular es el cuello de botella del algoritmo de Shor y en donde lógicamente mas esfuerzos se están haciendo para intentar mejorar la eficiencia del algoritmo.

En las secciones 4.4 y 4.5 se desarrollan algoritmos que implementan las compuertas cU_a a partir de compuertas fundamentales. En el proceso se requieren mas qubits ancilla, pero también se introducirá una ingeniosa técnica que permite ahorrar un gran cantidad de qubits en el primer registro.

4.3.4. Post-procesamiento Clásico: Expansión en fracciones continuas

Una vez hemos obtenido una fase φ de la estimación de fase, necesitamos deducir a partir de ella r para terminar de resolver el problema de hallar el orden. Como vimos en la sección anterior, $\varphi = j/r$ con $j = 0, \ldots, r-1$ si el cociente puede expresarse con t bits, mientras que es el mejor estimador con t cifras exactas en caso contrario. Pero sabemos que siempre j/r es un número racional por lo que si conseguimos encontrar el cociente de enteros más próximo podemos hallar r.

Las fracciones continuas son una forma de expresar números reales mediante una serie de enteros definidos de la siguiente forma

$$[a_0; \ldots, a_N] \equiv a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{\cdots + \cfrac{1}{a_N}}}} \tag{4.66}$$

donde $a_0, \ldots, a_N$ son enteros estrictamente positivos excepto a_0 que puede ser cero . Se define el *m-ésimo convergente* con $0 \leq m \leq N$ a esta fracción continua como $[a_0; \ldots, a_m]$. El algoritmo de expansión en fracciones continuas es un método para determinar la expansión en fracciones continuas de un número real arbitrario.

Sea $x \in \mathbb{R}$ el número del cuál queremos calcular su expansión en fracciones continuas, y sean i y f sus partes entera y no entera respectivamente. Entonces, se toma x y se le resta i, o sea $f = x - i$. Luego, si $f \neq 0$ se halla su inverso y se vuelve a repetir el paso anterior, es decir, $x = 1/f$. En caso contrario el algoritmo se detiene, lo que solo sucede si $x \in \mathbb{Q}$. Los coeficientes a_i están dados por el valor de i que se calcule en cada paso. Veamos un ejemplo práctico con $x = 2,4$:

1. $x = 2,4 \implies i = 2 \implies a_0 = 2$
2. $f = x - i = 2,4 - 2 = 0,4$
3. $x = 1/f = 1/0,4 = 2,5$
4. $x = 2,5 \implies i = 2 \implies a_1 = 2$
5. $f = x - i = 2,5 - 2 = 0,5$
6. $x = 1/f = 1/0,5 = 2$
7. $x = 2 \implies i = 2 \implies a_2 = 2$
8. $f = x - i = 2 - 2 = 0 \implies$ FIN

Por lo tanto, la expansión en fracciones continuas de $x = 2,4$ es

$$[2; 2, 2] \equiv 2 + \frac{1}{2 + \frac{1}{2}} = 2 + \frac{1}{2 + 0,5} = 2 + \frac{1}{2,5} = 2 + 0,4 = x \tag{4.67}$$

En este caso, como $x = 2,4 = 12/5$ es racional, la expansión en fracciones continuas es exacta y tiene fin. Probemos ahora calcular el convergente 3-ésimo de un número racional, por ejemplo $x = \pi$.

1. $x = \pi \implies i = 3 \implies a_0 = 3$
2. $f = x - i = \pi - 3$
3. $x = 1/f = 1/(\pi - 3) \sim 7,062$
4. $x \sim 7,06 \implies i = 7 \implies a_1 = 7$
5. $f = x - i \sim 0,062$
6. $x = 1/f \sim 16,13$
7. $x \sim 16,13 \implies i = 16 \implies a_2 = 16$
8. $f = x - i \sim 0,13$
9. $x = 1/f \sim 7,69$
10. $x \sim 7,69 \implies i = 7 \implies a_2 = 7$ Termino acá.

El convergente 3-ésimo de π es $[3; 7, 16, 7]$. La complejidad de calcular la expansión en fracciones continuas completa de un número racional $x = j/r$ donde j, r son enteros de n bits es $O(n^3)$, $O(n)$ pasos de 'separar e invertir' cada uno con $O(n^2)$ operaciones aritméticas. Por su parte, la complejidad de calcular el convergente m-ésimo de cualquier número real, racional o no, es $O(m^3)$.

Retomando el problema de hallar el orden, la expansión en fracciones continuas nos proporciona ciertos racionales j', r' sin factores comunes cuyo cociente es igual a j/r, por lo que r' es un candidato a ser el orden que hallamos buscando. Esto lo podemos comprobar fácilmente haciendo $a^{r'} \bmod N$ y de ser 1 el resultado, hemos resuelto el problema. Luego, como dijimos en la sección 4.3.2, para culminar el algoritmo de Shor se calcula $\text{gdc}(a^{r'/2} \pm 1, N)$ y a menos que r' sea impar o $a^{r/2} \equiv -1 \bmod N$, hemos hallado los dos factores primos de N, p y q.

4.4. Implementación con 2n+3 qubits del Algoritmo de Shor

En la sección 4.3 nunca se especificó como implementar las compuertas U_a, si no que se dejaron como como cajas negras ya que no era necesario para el funcionamiento general del algoritmo de Shor. Sin embargo, si queremos realmente implementar numéricamente el algoritmo de forma

completa, debemos construir una compuerta cU_a que sea reversible y mas aún, deducirla a partir de compuertas fundamentales. En esta sección nos centraremos en la implementación desarrollada por Stéphane Beauregard en [3], la cuál requiere de $2n+3$ qubits para factorizar un número N de n bits. Aunque para implementar la exponenciación modular reversible se requieran más qubits ancilla, en la subsección 4.4.6 se desarrolla una alternativa a la QFT que en vez de requerir t qubits para el primer registro, solo requiere uno; de aquí que la cantidad de qubits necesarios solo dependa del tamaño del número y no de la cantidad de cifras exactas al estimar la fase. Además, el circuito requiere $O(n^3 \log n)$ compuertas cuánticas elementales y es general ya que no depende de las propiedades específicas del N que deseemos factorizar. El circuito de S. Beauregard a su vez se inspira en parte en el presentado por Vedral, Barenco y Ekert en [10].

4.4.1. ϕ Adder

Obviamente la primera operación algebraica que se debe implementar es la suma, al ser esta la base de todas las demás. Existen algoritmos con acarreo, análogos a los clásicos como el circuito 2.3 y 2.4, pero es importante notar que no es necesario un algoritmo para sumar dos números *cuánticos*. Como en la exponenciación modular $a^x \bmod N$ tanto $a < N$ son números enteros definidos, y sólo x representa un número cuántico, solo necesitamos sumar un número clásico a uno cuántico. Por número cuántico hacemos referencia a un registro $|b\rangle$ de n qubits que puede estar en cualquier superposición de sus 2^n estados, por lo que $a^x \bmod N$ es en realidad un registro que contiene una superposición de estados con $x = 0, \ldots, 2^n - 1$. Luego,

$$|b\rangle = \sum_{i=0}^{2^n-1} b_i \, |i\rangle \tag{4.68}$$

Por lo tanto, la suma reversible de los registros $|b\rangle \otimes |a\rangle \rightarrow |b\rangle \otimes |b+a\rangle$ es la superposición de la suma de cada estado de $|b\rangle$ con cada estado de $|a\rangle$, todo modulo 2^n al tener ambos registros n qubits.

$$|a+b\rangle \propto \sum_{k=0}^{2^n-1} \sum_{i,j \;\mid\; (i+j) \bmod 2^n = k} (a_j + b_i) \, |k\rangle \tag{4.69}$$

Por el contrario, sumar un número clásico a a uno cuántico $|b\rangle$ se corresponde a la transformación $|b\rangle \rightarrow |a+b\rangle$. Como vemos no requiere dos registros, y además, como podemos saber de antemano el valor de a, podemos construir las compuertas de modo que ya incluyan el valor particular de a antes de ejecutar la compuerta de suma. Entonces

$$|a+b\rangle = \sum_{i=0}^{2^n-1} b_i \, |(i+a) \bmod 2^n\rangle \tag{4.70}$$

Hecha esta aclaración sobre la diferencia entre números cuánticos y clásicos, por simplicidad en la explicación vamos a asumir sin perdida de generalidad que el registro $|b\rangle$ no está superpuesto, es decir, que solo b_b es distinto de cero. Esto no equivale a decir que sea un número clásico, y además, podemos recuperar el caso general fácilmente gracias a la linealidad del Álgebra al considerar la suma de todos los casos individuales de $b = 0, \ldots, 2^n - 1$.

En [18], Draper introduce un algoritmo para la suma modulo 2^n de dos registros cuánticos $|a\rangle$ y $|b\rangle$ de n qubits en el espacio de Fourier, que no necesita qubits ancilla. (Circuito 4.7)

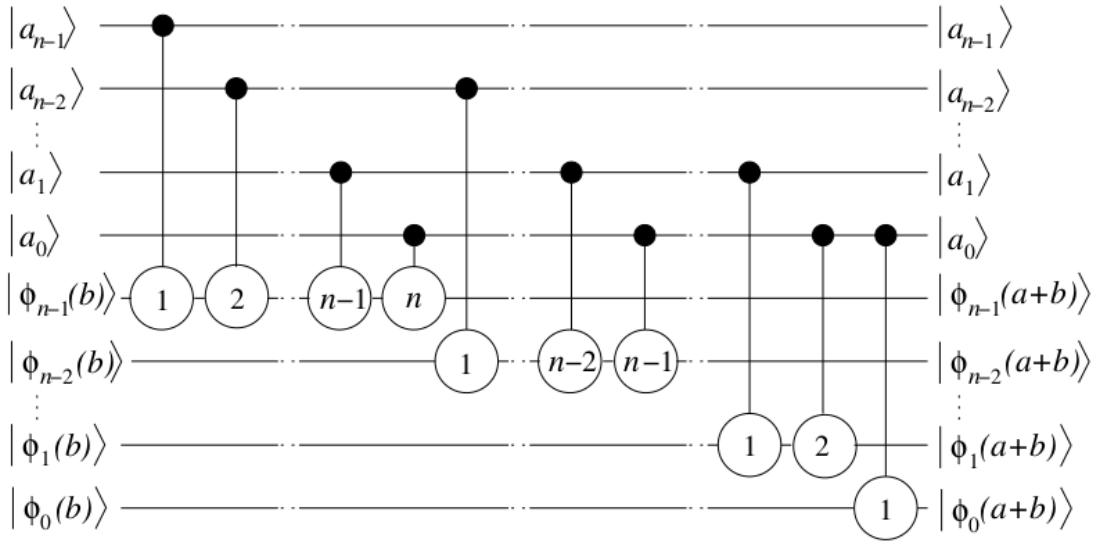


Figura 4.7: Algoritmo de Draper para sumar dos registros cuánticos

Aquí, $|\phi_i(b)\rangle$ corresponde al j-ésimo qubit del resultado de aplicar la QFT a $|b\rangle$. En nuestro caso, como a es un número clásico, no necesitamos los qubits $|a_j\rangle$ por lo que las rotaciones controladas cR_k serán en cambio rotaciones R_k que solo serán ejecutadas en caso de que el bit correspondiente de a, es decir a_{n-k}, sea un 1. Llamaremos esta compuerta $\phi ADD(a)$ ya que se lleva a cabo en el espacio de Fourier. Entonces, para sumarle a al registro $|b\rangle$

$$|b\rangle \implies \begin{array}{c} \\ \vdots \\ \\ \\ \end{array} \boxed{QFT} \boxed{\begin{array}{c}\phi\\ \text{A}\\ \text{D}\\ \text{D}\\ \text{(a)}\end{array}} \boxed{QFT^{-1}} \begin{array}{c} \\ \vdots \\ \\ \\ \end{array} \implies |a+b\rangle \tag{4.71}$$

Como el registro $|b\rangle$ tiene n qubits, la salida es en realidad $|(a+b) \bmod 2^n\rangle$, por lo que debemos agregarle un qubit si no queremos perder el qubit mas significativo de la suma. Es decir, $|0\rangle \otimes |b\rangle \rightarrow$

$|a+b\rangle$. Al ser una compuerta reversible, $\phi ADD(a)^{-1} = \phi ADD(-a)$ por lo que aplicándola a una entrada $|\phi(b)\rangle$ podemos obtener la operación de sustracción modular

$$\phi ADD(a)\,|\phi(b)\rangle = \begin{cases} \phi(b-a) & \text{si } b \geq a \\ \phi(2^{n+1} - (a-b)) & \text{si } b < a \end{cases} \tag{4.72}$$

Por lo tanto, si $b < a$ el qubit mas significativo de la resta va a ser siempre 1, y viceversa. Este hecho es útil a la hora de implementar la suma modular, ya que permite comparar a y b.

4.4.2. Adición Modular

Con la compuerta $\phi ADD(a)$ y la QFT, podemos construir un circuito para computar la adición modulo N, que denotaremos como $\phi ADD(a)$ mod N y es el elemento fundamental de esta sección. Básicamente, la idea es comparar $a+b$ y N, y luego sumar al registro $|b\rangle$ lo que corresponda, $a-N$ si $a+b \geq N$ o a si $a+b < N$. Aunque parezca una operación sencilla de implementar, el problema esta en lograr hacerlo de una manera reversible, o sea, revirtiendo los qubits de trabajo a su estado inicial una vez computada la suma.

El circuito requiere un qubit ancilla más, además del necesario para $\phi ADD(a)$. También, se incluyen por su posterior necesidad dos qubits de control. En el circuito 4.8 se muestra la secuencia de compuertas, que se explica a continuación. El estado inicial del sistema es

$$|c_1\rangle \otimes |c_2\rangle \otimes |b\rangle \otimes |0\rangle \tag{4.73}$$

donde en el caso general el registro $|b\rangle$ cumple que, como $b < N$ entonces

$$|b\rangle = \sum_{i=0}^{N-1} b_i\,|i\rangle \tag{4.74}$$

Luego, se aplica la QFT al registro $|b\rangle$, y se le suma a, controlado por los qubits $|c_1\rangle \otimes |c_2\rangle$.

$$\phi ADD(a)\,|\phi(b)\rangle = |\phi(b+a)\rangle \tag{4.75}$$

Ahora se resta N para compararlo con $a+b$. Esta compuerta no es controlada, es decir, siempre se ejecuta sin importar $|c_1\rangle \otimes |c_2\rangle$. Entonces, vemos que

$$\begin{aligned} \phi ADD(N)\,|\phi(b+a)\rangle &= |\phi(a+b-N) \bmod 2^{n+1}\rangle \\ &= \begin{cases} |\phi((a+b-N) \bmod 2^{n+1})\rangle = |\phi(2^{n+1} - (N-(a+b)))\rangle & \text{si } a+b < N \\ |\phi(a+b-N)\rangle & \text{si } a+b \geq N \end{cases} \end{aligned} \tag{4.76}$$

Como $a, b < N$ el segundo caso implica también $a + b - N < b$. Para poder extraer el qubit más significativo, que será $|q\rangle = |1\rangle$ si $a + b < N$ o $|q\rangle = |0\rangle$ si no, se aplica la QFT^{-1} y se usa una compuerta CNOT controlada por $|q\rangle$ sobre el qubit ancilla que estaba en cero. Luego, se vuelve el registro al espacio de Fourier con otra QFT.

$$\text{CNOT}_{1\ 2}\,|(a+b-N) \bmod 2^{n+1}\rangle \otimes |0\rangle = \begin{cases} |2^{n+1} - (N - (a+b))\rangle \otimes |1\rangle & \text{si } a+b < N \\ |a+b-N\rangle \otimes |0\rangle & \text{si } a+b \geq N \end{cases} \tag{4.77}$$

Ahora, controlando sólo con el ancilla se suma N, por lo que sólo en el caso $a+b < N$ se tiene que

$$\begin{gathered} 2^{n+1} - (N - (a+b)) + N = 2^{n+1} + (a+b) > 2^{n+1} \\ \implies 2^{n+1} - (N - (a+b)) + N) \bmod 2^{n+1} = 2^{n+1} + (a+b) - 2^{n+1} = a+b \end{gathered} \tag{4.78}$$

Entonces, obtenemos para $\phi((a+b) \bmod N$ para ambos casos

$$\begin{aligned} &C\phi ADD(N)_{2\ 1}\,|(a+b-N \bmod 2^{n+1}\rangle \otimes |q\rangle \\ &= \begin{cases} |\phi(a+b)\rangle \otimes |1\rangle & \text{si } a+b < N \\ |\phi(a+b-N)\rangle \otimes |0\rangle & \text{si } a+b \geq N \end{cases} \end{aligned} \tag{4.79}$$

Ahora solo queda restablecer el qubit ancilla a $|0\rangle$, que en el caso $a+b < N$ ha sido cambiado a $|1\rangle$. Como el algoritmo incluye transformaciones de Fourier, este restablecimiento no es simplemente recorrer de forma inversa la secuencia de compuertas. Pero usando la siguiente identidad

$$(a+b) \bmod N \geq a \iff a+b < N \tag{4.80}$$

podemos repetir la misma idea de obtener el bit más significante de una comparación, en este caso comparando el registro $|(a+b) \bmod N\rangle$ con a. Restamos a controlando con los dos primero qubits, por lo que si $a+b \geq N$ tenemos que $|\phi((a+b-N-a) \bmod 2^{n+1})\rangle = |\phi(2^{n+1} - (N-b))\rangle$

$$\phi ADD(a)\,|(a+b)\text{mod } N\rangle \otimes |q\rangle = \begin{cases} |\phi(b)\rangle \otimes |1\rangle & \text{si } a+b < N \\ |\phi(2^{n+1} - (N-b))\rangle \otimes |0\rangle & \text{si } a+b \geq N \end{cases} \tag{4.81}$$

Vemos que ahora el registro tiene el qubit mas significativo en 1 si y sólo si $a+b \geq N$, por lo que si lo negamos, vamos a tener al qubit en 1 si y sólo si $a+b < N$. Para esto, se aplica la QFT^{-1} para sacar el registro del espacio de Fourier y se aplica un NOT sobre el qubit más significativo, o sea, se suma $2^n \bmod 2^{n+1}$

$$\begin{cases} |b\rangle \to |(b+2^n) \bmod 2^{n+1}\rangle = |2^n + b\rangle & \text{si } a+b < N \\ |2^{n+1} - (N-b)\rangle \to |(2^{n+1} - (N-b) + 2^n) \bmod 2^{n+1}\rangle = |2^n - (N-b)\rangle & \text{si } a+b \geq N \end{cases} \tag{4.82}$$

Vemos que como $b < N \implies 0 < N - b < 2^n$, entonces $2^{n+1} - (N-b) + 2^n > 2^{n+1}$ y por lo tanto

$$(2^{n+1} - (N-b) + 2^n) \bmod 2^{n+1} = 2^n - (N-b) \tag{4.83}$$

Luego, se aplica un CNOT controlado por este sobre el qubit ancilla. La compuerta NOT sobre el qubit mas significativo se necesita porque queremos negar el qubit ancilla sólo en el caso en que haya sido alterado, o sea, $a + b < N$.

$$\begin{cases} \text{CNOT}_{1\ 2}\,|2^n + b\rangle \otimes |1\rangle = |2^n + b\rangle \otimes |0\rangle & \text{si } a + b < N \\ \text{CNOT}_{1\ 2}\,|2^n - (N-b)\rangle \otimes |0\rangle = |2^n - (N-b)\rangle \otimes |0\rangle & \text{si } a + b \geq N \end{cases} \tag{4.84}$$

Una vez revertido el qubit ancilla a $|0\rangle$, se aplica la secuencia inversa de compuertas que se usaron para la reversión, que como no dependían del qubit ancilla devuelve el estado del registro a $|(a+b) \bmod N\rangle$ en el registro. Como corresponde para que sea reversible, el estado final es

$$|c_1\rangle \otimes |c_2\rangle \otimes |(a+b) \bmod N\rangle \otimes |0\rangle \tag{4.85}$$

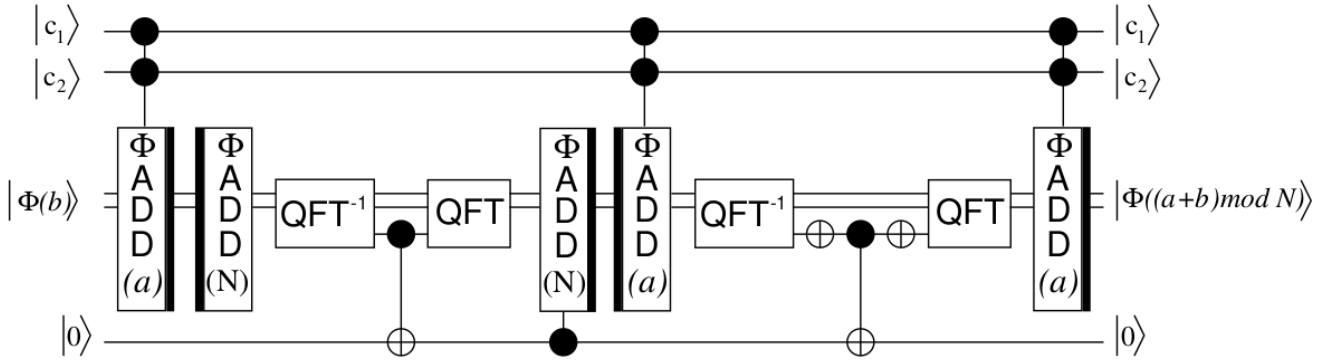


Figura 4.8: Circuito doblemente controlado para implementar la adición modulo N usando las compuertas de adición $\phi ADD(a)$

Puede verse por simple inspección del circuito 4.8 que si c_2 o c_2 son nulos, el estado final del sistema es el mismo que el inicial, a pesar de haberse ejecutado varias compuertas que no son controladas por los dos primeros qubits. Esto facilita la implementación del algoritmo, ya que no hay compuertas con más de dos controles que serían complicadas de lograr.

4.4.3. Multiplicación Modular

Una vez ya tenemos la adición modular, el siguiente paso, la multiplicación modular a la cuál llamaremos $CMULT(a) \bmod N$, es bastante directo. Esta compuerta tiene como entrada $|c\rangle \otimes |x\rangle \otimes |b\rangle \otimes |0\rangle$ donde $|c\rangle$ es un qubit de control y $|x\rangle$ y $|b\rangle$ son registros de n y $n+1$ qubits

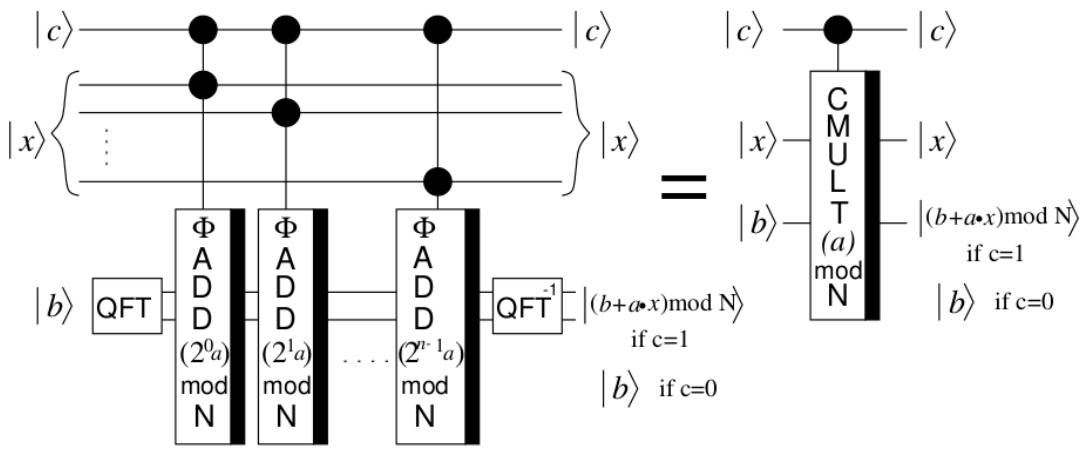


Figura 4.9: Circuito para implementar la compuerta $CMULT(a) \bmod N$ a partir de $\phi ADD(a) \bmod N$.

respectivamente, además del qubit ancilla $|0\rangle$. Como vemos, ya tenemos los $2n+3$ qubits que le dan nombre a este algoritmo, por lo que a partir de ahora no le agregaremos más.

Si $|c\rangle = |1\rangle$, la salida es $|c\rangle \otimes |x\rangle \otimes |b + (ax) \bmod N\rangle \otimes |0\rangle$, mientras que si $|c\rangle = |0\rangle$, la salida es igual a la entrada. Usando que

$$(ax) \bmod N = ((\ldots(((x_0 2^0 a) \bmod N) + x_1 2^1 a) \bmod N) + \cdots + x_{n-1} 2^{n-1} a) \bmod N \qquad (4.86)$$

podemos ver que la multiplicación modular son sólo sumas modulares anidadas. En particular, para calcular $(ax) \bmod N$ con $x = \sum x_i 2^i$ hay que sumar en cada paso $x_i 2^i$ modulo N al registro $|b\rangle$ para obtener $|(b + x_i 2^i) \bmod N\rangle$. En el circuito 4.9 se muestra la implementación de $CMULT(a) \bmod N$ a partir de $\phi ADD(a) \bmod N$.

Logramos construir una compuerta controlada que nos permite hacer la transformación $|x\rangle \otimes |b\rangle \rightarrow |x\rangle \otimes |b + (ax) \bmod N\rangle$. Como la multiplicación modular es sólo un paso previo para la exponenciación modular, necesitamos más bien lograr $|x\rangle \rightarrow |(ax) \bmod N\rangle$, es decir, $b = 0$.

4.4.4. U_a y Exponenciación Modular

Usando dos compuertas de multiplicación modular podemos construir finalmente U_a, cuya acción es $U_a |x\rangle = |(ax) \bmod N\rangle$. Primero, usando el circuito de la sección anterior, aplicamos $CMULT(a) \bmod N$ a una entrada $|c\rangle \otimes |x\rangle \otimes |0\rangle$ y obtenemos $|c\rangle \otimes |x\rangle \otimes |(ax) \bmod N\rangle$. Luego, usamos compuertas SWAP para intercambiar el primer y el segundo registro. En la sección anterior dijimos que $|b\rangle$ tenia $n+1$ qubits, pero si prestamos atención, podemos ver que el qubit $n+1$ que es necesario para almacenar el qubit más significativo es en realidad un qubit ancilla, ya que

al aplicar $CMULT(a)$ mod N siempre vuelve a su estado inicial $|0\rangle$. Entonces, intercambiamos el registro $|x\rangle$ con los n primeros qubits del segundo registro, obteniendo $|c\rangle \otimes |(ax) \text{ mod } N\rangle \otimes |x\rangle$.

Ahora, debemos encontrar la forma de limpiar el segundo registro, o sea, necesitamos hacer la transformación $|c\rangle \otimes |(ax) \text{ mod } N\rangle \otimes |x\rangle \rightarrow |c\rangle \otimes |(ax) \text{ mod } N\rangle \otimes |0\rangle$. Para lograr esto, vamos a aplicar la inversa compuerta $CMULT(a)$ mod N. Este a^{-1} es el inverso multiplicativo de a modulo N, o sea $a^{-1}a \equiv 1 \text{mod } N$, y es computable en tiempo polinómico, además de siempre existir ya que a y N son coprimos. En el circuito 4.10 se muestra esquemáticamente la construcción de U_a.

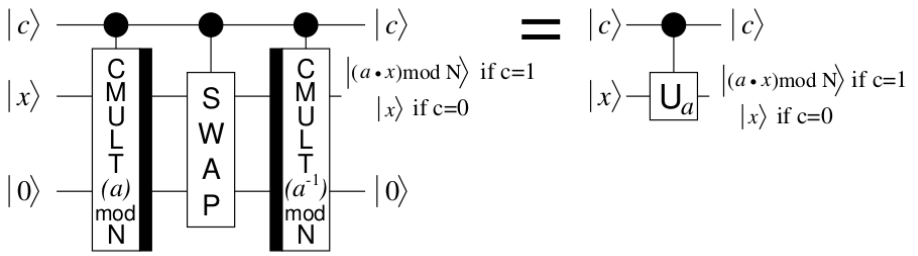


Figura 4.10: Circuito para implementar la compuerta cU_a controlada con un qubits, en base a las compuerta de multiplicación modular $CMULT(a)$ mod N y $CMULT(a^{-1})$ mod N.

Entonces, U_a realiza la siguiente secuencia de transformaciones en los dos registros

$$\begin{aligned} |x\rangle \otimes |0\rangle \rightarrow\ & |x\rangle \otimes |(ax) \text{ mod } N\rangle \\ \rightarrow\ & |(ax) \text{ mod } N\rangle \otimes |x\rangle \\ \rightarrow\ & |(ax) \text{ mod } N\rangle \otimes |(x - a^{-1}ax) \text{ mod } N\rangle \\ \rightarrow\ & |(ax) \text{ mod } N\rangle \otimes |0\rangle \end{aligned} \tag{4.87}$$

Como a es clásico, si queremos multiplicar por a^k no es necesario (aunque posible) aplicar k veces la compuerta U_a, si no que podemos simplemente aplicar U_{a^k}, donde a^k mod N es precomputada clásicamente. Esto es porque

$$(a^k x) \text{ mod } N = \underbrace{(a \dots (a(ax) \text{ mod } N) \text{ mod } N) \dots) \text{ mod } N}_{k \text{ veces}} \tag{4.88}$$

Si queremos entonces calcular $|a^x \text{ mod } N\rangle$ donde x es un número cuántico, notemos que podemos descomponer $x = \sum_{i=0}^{n-1} x_i 2^i$. Luego

$$\begin{aligned} |a^x \text{ mod } N\rangle &= |a^{\sum_{i=0}^{n-1} x_i 2^i} \text{ mod } N\rangle \\ &= |\prod_{i=0}^{n-1} a^{x_i 2^i} \text{ mod } N\rangle \end{aligned} \tag{4.89}$$

Vemos que si $x_j = 1$ entonces el qubit j-ésimo del registro $|x\rangle$ es $|1\rangle$, es decir

$$|x\rangle = |x_{n-1}x_{n-2}\ldots, \underset{j\text{-ésimo}}{1} \ldots x_0\rangle \tag{4.90}$$

Luego, podemos usar compuertas controladas para lograr la expresión (4.89). Podemos verlo más claramente usando inducción: Si aplicamos la compuerta $U_{a^{2^0}} = U_a$ al primer registro inicializado a $|1\rangle$, entonces obtenemos

$$U_a\,|1\rangle = |a \bmod N\rangle \tag{4.91}$$

Asumiendo que hemos transformado correctamente el primer registro hasta alcanzar el estado $|\prod_{i=0}^{j} a^{x_i 2^i} \bmod N\rangle$, si ahora aplicamos la compuerta $U_{a^{2^{j+1}}}$ controlada en el qubit $(j+1)$-ésimo del primer registro, tenemos

$$\begin{aligned} U_{a^{2^{j+1}}}\,|\prod_{i=0}^{j} a^{x_i 2^i} \bmod N\rangle &= |\big((\prod_{i=0}^{j} a^{x_i 2^i})\; a^{2^{j+1}}\big) \bmod N\rangle \\ &= |\prod_{i=0}^{j+1} a^{x_i 2^i} \bmod N\rangle \end{aligned} \tag{4.92}$$

Repitiendo hasta aplicar $U_{a^{2^n}}$, obtenemos al final en el registro

$$|\prod_{i=0}^{n} a^{x_i 2^i} \bmod N\rangle = |a^x \bmod N\rangle \tag{4.93}$$

Ya tenemos la compuerta U_a para llevar a cabo la estimación de fase, como en el circuito 4.6. Es por esto que a lo largo de todo el desarrollo de este algoritmo hemos dejado un qubit de control $|c\rangle$. Cada compuerta $cU_{a^{2^j}}$ es controlada por el qubit j-ésimo del registro de t qubits para la estimación de fase. Pasaremos a llamar a este primer registro, mientras que al registro que contiene $|a^x \bmod N\rangle$ lo dejamos como segundo. Luego, si t es igual a n, habremos obtenido aplicar la estimación una superposición de $|a^x \bmod N\rangle$ con $x = 0, \ldots, 2^n - 1$ en el segundo registro.

Como vimos que $U_a^{2^n} = U_{a^{2^n}}$, el circuito completo para aplicar el algoritmo de Shor, usando las compuertas U_a que hemos deducido hasta ahora en esta sección, esta dado por el circuito 4.11 y usa $n+2$ qubits ancilla y un total de $t + 2n + 2$ qubits.

4.4.5. Transformada de Fourier Semiclásica

Posiblemente el avance más importante de esta implementación es la aplicación de la Transformada de Fourier *Semiclásica* para reducir el número de qubits del primer registro de t a sólo uno. En [19], Robert B. Griffiths y Chi-Sheng Niu demostraron que es posible llevar a cabo la transformada inversa de Fourier de la estimación de fase de una forma semiclásica al usar la señal resultante de medir cada qubit para determinar como medir los siguientes. De esta forma, todas las

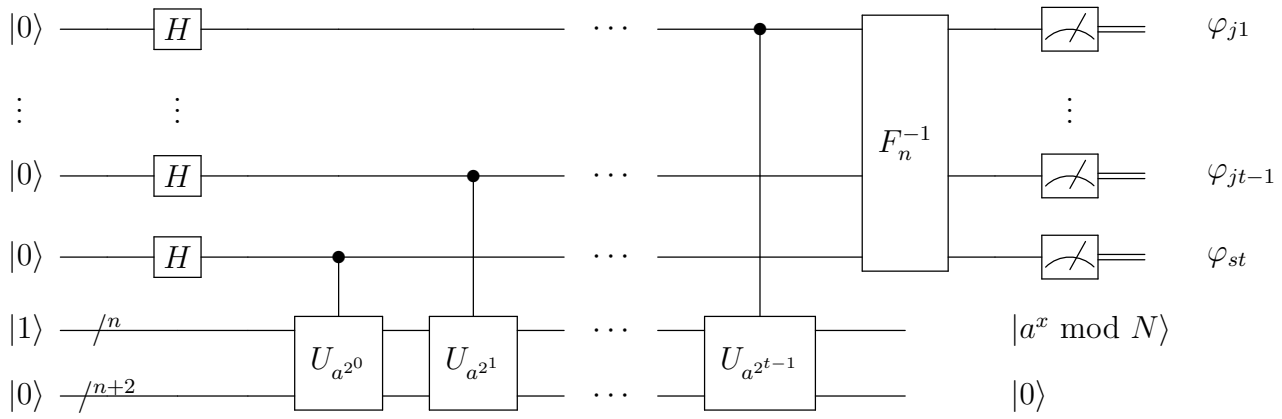


Figura 4.11: Circuito Cuántico para implementar el algoritmo de Shor con $t+2n+2$ qubits, donde x es un número cuántico superpuesto en $x = 0, \ldots, 2^t - 1$.

compuertas de rotación controlada de la QFT^{-1} se reemplazan por rotaciones de un qubit, donde la fase de la rotación depende del resultado de las mediciones anteriores. Esta implementación preserva las probabilidades al medir la fase $\varphi_s \sim s/r$, que es el objetivo de la estimación de fase.

Repasando el circuito 4.4, gracias a la posibilidad de intercambiar el qubit de control y acción en las rotaciones controladas, podemos modificar la última parte del algoritmo de Shor como se muestra en el circuito 4.12

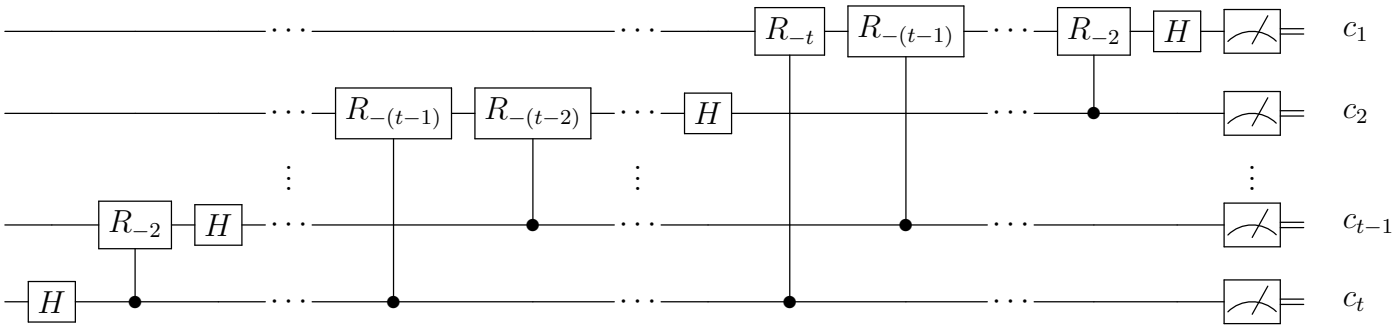


Figura 4.12: Circuito que muestra las últimas operaciones sobre el primer registro en el algoritmo de Shor, la compuerta QFT^{-1} y la medición de cada qubit. Aquí c_j son el resultado de medir cada bit de la fase $\varphi_s \sim s/r$.

.

Supongamos que medimos $c_t = 1$. Como el qubit t-ésimo sólo fue modificado por la compuerta de Hadamard, podemos concluir que desde ese momento se encuentra en el mismo estado, o sea $|1\rangle$. Pero luego de la compuerta de Hadamard, este qubit es usado para controlar la rotación cR_{-2}

sobre el qubit $t-1$. Entonces, podemos también concluir que como el qubit de control estaba en el estado $|1\rangle$, la acción de cR_{-2} es la misma que si fuera una rotación no controlada R_{-2}. En caso de haber medido $c_t = 0$, nuestro razonamiento nos lleva a que el qubit de control estaba en el estado $|0\rangle$ y que entonces la compuerta cR_{-2} no se ejecuta y no se cambia el estado del qubit $t-1$.

Siguiendo el mismo razonamiento, el qubit $t-1$ es modificado por última vez en la compuerta de Hadamard inmediatamente posterior a esta rotación, por lo que puedo asumir que luego de esta compuerta el qubit estaba en el mismo estado que inmediatamente antes de la medición. Entonces, en todas las rotaciones en las cuáles el $(t-1)$-ésimo qubit es el control, dicha rotación se va a producir o no según el resultado de la medición haya sido $c_{t-1} = 1$ o 0, respectivamente. Veamos un ejemplo con los dos último qubits de que las probabilidades de medición se conservan. Primero midiendo al final, como hacíamos hasta ahora:

$$\begin{aligned} H_2 |+\rangle |0\rangle = |+\rangle |+\rangle \to R_{21\ -2} |+\rangle |+\rangle &= \frac{1}{2}(|0\rangle + |1\rangle + |2\rangle + R_{-2} |3\rangle) \\ &= \frac{1}{2}(|0\rangle + |1\rangle + |2\rangle + e^{\pi i/2} |3\rangle) \to \frac{1}{2} H_1(|0\rangle + |1\rangle + |2\rangle + i |3\rangle) \\ &\to \frac{1}{2\sqrt{2}}(|+,0\rangle + |+,1\rangle + |-,0\rangle + i |-,1\rangle) \\ &= \frac{1}{2\sqrt{2}}(|0\rangle + |1\rangle + |2\rangle + |3\rangle + |0\rangle + i |1\rangle - |2\rangle - i |3\rangle) \\ &= \frac{1}{2\sqrt{2}}(2 |0\rangle + (1+i) |1\rangle + (1-i) |3\rangle) \end{aligned}$$

$$P(0) = 1/2 \quad P(1) = 1/4 \quad P(2) = 0 \quad P(3) = 1/4 \tag{4.94}$$

Si mido el segundo qubit inmediatamente después de H_2

$$\begin{aligned} H_2 |+\rangle |0\rangle = |+\rangle |+\rangle \to M_2 |+\rangle |+\rangle \quad &\text{mido el segundo qubit} \to P_2(0) = 1/2 = P_2(1) \\ = &\begin{cases} |+\rangle |0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |2\rangle) & \text{si } c = 0 \\ |+\rangle |1\rangle = \frac{1}{\sqrt{2}}(|1\rangle + |3\rangle) & \text{si } c = 1 \end{cases} \\ \to &\begin{cases} \frac{1}{\sqrt{2}} R_{21\ -2}(|0\rangle + |2\rangle) = \frac{1}{\sqrt{2}} |0\rangle + |2\rangle = & \text{si } c = 0 \\ \frac{1}{\sqrt{2}} R_{21\ -2}(|1\rangle + |3\rangle) = \frac{1}{\sqrt{2}} |1\rangle + i |3\rangle & \text{si } c = 1 \end{cases} \\ \to &\begin{cases} H_1 |+,0\rangle = |0,0\rangle = |0\rangle & \text{si } c = 0 \\ \frac{1}{\sqrt{2}} H_1(|1\rangle + i |3\rangle) = \frac{1}{\sqrt{2}}(|+,1\rangle + i |-,1\rangle) = \frac{1}{2}((1+i) |1\rangle + (1-i) |3\rangle) & \text{si } c = 1 \end{cases} \end{aligned}$$

$$P(0) = 1/2 \quad P(1) = 1/4 \quad P(2) = 0 \quad P(3) = 1/4 \tag{4.95}$$

Se puede demostrar que esto vale en general. Prosiguiendo con esta idea, se pueden mover todas las mediciones inmediatamente después de las compuertas de Hadamard que actúan sobre cada

qubit en la QFT^{-1}, y luego reemplazar las cR_{-k} por rotaciones controladas clásicamente. A partir del circuito 4.12, se llega a:

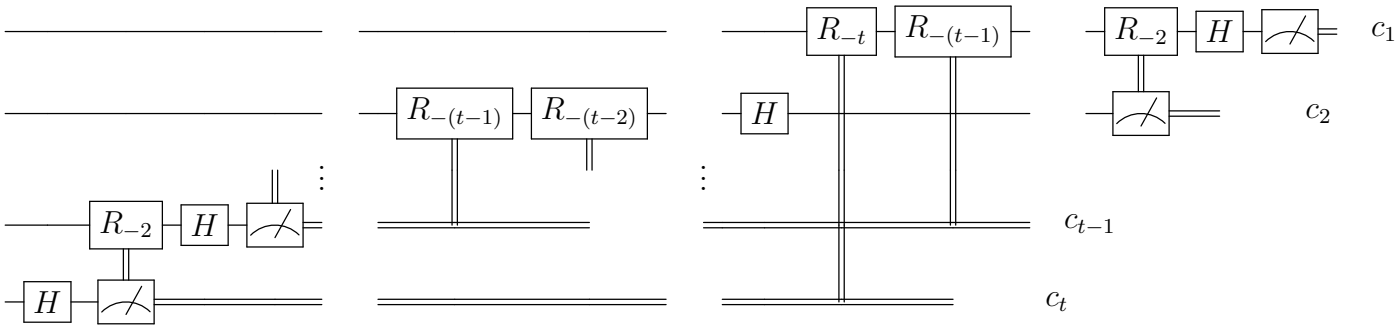


Figura 4.13: Circuito de la compuerta QFT^{-1} donde se han reemplazado las cR_{-k} por rotaciones clásicas controladas por el resultado de mediciones anteriores. La doble línea simboliza señales clásicas.

Se observa que no hay ninguna compuerta de dos qubits ni compuertas que se ejecuten de forma paralela, ya que las mediciones proporcionan una señal clásica a las rotaciones para saber si se aplican o no.

Usando esto en el circuito 4.11, vemos allí que luego de controlar la compuerta $cU_{a^{2^{t-j}}}$, el qubit j-ésimo del primer registro no vuelve a intervenir hasta la QFT^{-1}, por lo que podemos adelantar las compuertas que actúan sobre él inmediatamente después de la compuerta $cU_{a^{2^{t-j}}}$, como se hace en el circuito 4.14.

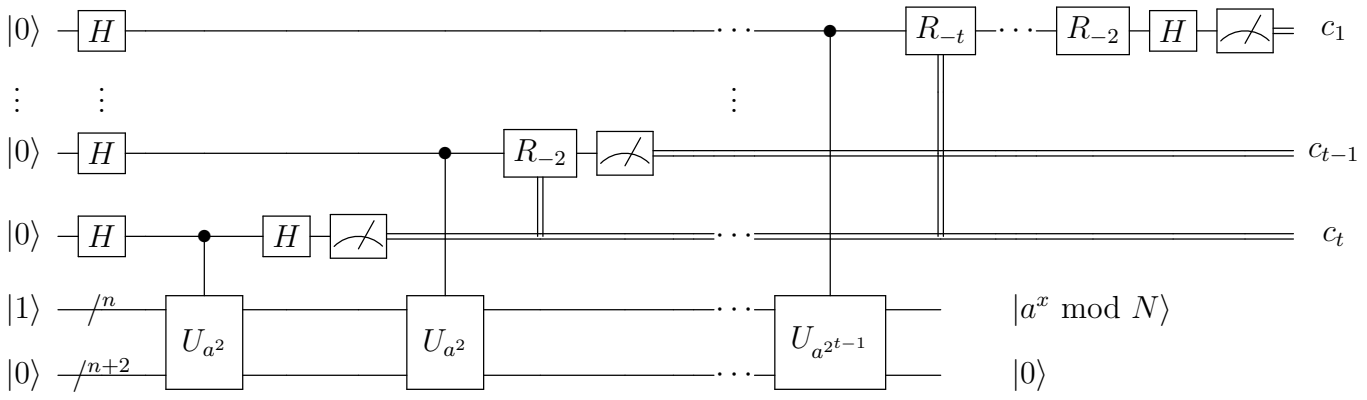


Figura 4.14: Aplicación de la transformada semiclásica de Fourier al circuito del algoritmo de Shor con $t + 2n + 2$ qubits.

Nada nos impide el siguiente paso: usar un sólo qubit. Si el resultado de cada medición c_j se almacena, luego puede ser usado para controlar la ejecución o no de rotaciones posteriores. Sólo es necesario agregar compuertas NOT inmediatamente después de cada medición, controladas por el resultado de estas. Esto es para devolver el qubit al estado $|+\rangle$ que es el estado que tiene que tener a la hora de controlar las compuertas $cU_{a^{2^j}}$. En el circuito 4.15 se muestra el resultado.

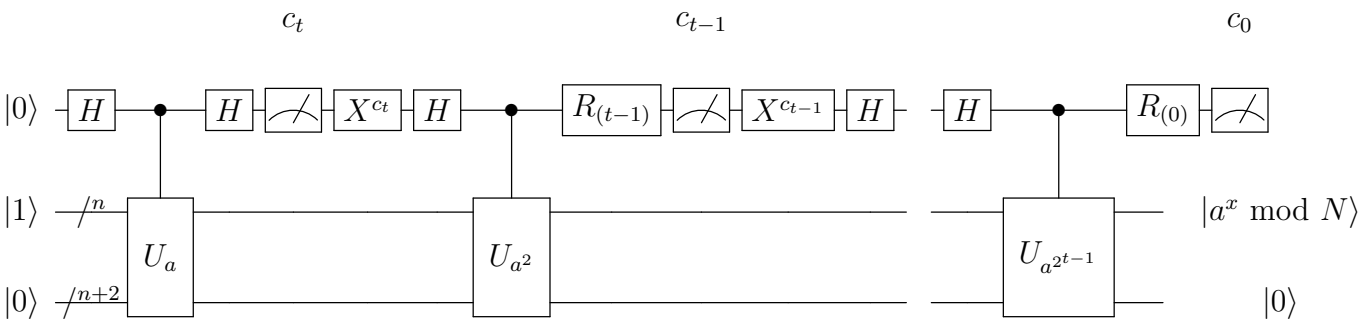


Figura 4.15: Circuito Cuántico para implementar el algoritmo de Shor con $2n + 3$ qubits. Las compuertas de $R_{(j)}$ son rotaciones que depende de todas las mediciones c_k con $k > j$.

4.4.6. QFT aproximada

En el artículo de Beauregard, [3], se desarrolla una forma de simplificar la QFT al descartar rotaciones que sean más pequeño que cierto umbral de error, ya que dichas fases serían absorbidas por el error. También, en las compuertas de $\phi\ ADD$ sucede lo mismo, ya que ambas compuertas funcionan de manera similar. Por definición, a medida que crece n QFT y $\phi\ ADD$ realiza rotaciones con fase cada vez más pequeñas, ya que la fase es proporcional a la distancia entre el índice del qubit de control y el del qubit de acción. Tanto para simulaciones como para implementaciones físicas, existe cierto k_{max} tal que para toda R_k con $|k| > k_{max}$ la fase es despreciable al quedar esta absorbida por el error. Podemos entonces ignorar estas rotaciones con $|k| > k_{max}$, y se tiene que estas QFT aproximadas son cercanas la exacta incluso para k_{max} logarítmico en n. Puede demostrarse, [16], que el error de hacer esto es proporcional a $n2^{-k_{max}}$, por lo que podemos tomar $k_{max} = O(\log(n/\epsilon))$.

4.4.7. Análisis de Complejidad

Suponiendo queremos factorizar N de n bits, ahora analizaremos la complejidad computacional del circuito que hemos desarrollado en esta sección, teniendo en cuenta el número de qubits, la cantidad de compuertas y la profundidad algorítmica del circuito. Con respecto a la profundidad,

vamos a considerar que se pueden llevar a cabo simultáneamente compuertas sobre diferentes qubits, pero es imposible que un mismo qubit sirva como control en dos compuertas al mismo tiempo. En cuanto a compuertas fundamentales, el circuito usa compuertas de un sólo qubit, y luego compuertas de rotación y negación con hasta dos qubits de control. Estas últimas pueden ser implementadas con un número constante de compuertas controladas de un qubit y CNOTs, como ya vimos en la subsección 3.2.3.

La compuerta $\phi ADD(a)$ desarrollada en la sección 4.4.1 requiere $n+1$ qubits y $O(n)$ compuertas de un solo qubit, con una profundidad algorítmica constante. Sin embargo, como luego en $\phi ADD(a)$ mod N se usan compuertas de adición controladas, las rotaciones del circuito 4.7 tienen que hacerse de forma secuencial, ya que el qubit de control sólo puede controlar una a la vez. Sólo una de las adiciones no tiene ningún qubit de control.

La compuerta $\phi ADD(a)$ mod N, circuito 4.8, requiere por sí misma $n+4$ qubits. La implementación de la QFT aproximada en $n+1$ qubits requiere $O(nk_{max})$ compuertas, y no se conoce manera de mejorar la profundidad mas allá de $O(n)$ sin usar qubits ancilla. Entonces, la adición modular requiere $O(nk_{max})$ compuertas y una profundidad $O(n)$. La multiplicación modular $CMULT(a)$ mod N son simplemente n compuertas de adición modular más ir y volver al espacio de Fourier, por lo que requiere $2n+3$ qubits, $O(n^2k_{max})$ compuertas y una profundidad $O(n^2)$. Dos multiplicaciones modular más el intercambio de los registros (4.10) forman la compuerta U_a. Para realizar el intercambio son necesarias $O(n)$ compuertas con profundidad tambien $O(n)$, por lo que el circuito U_a completo requiere también $2n+3$ qubits, $O(n^2k_{max})$ compuertas y una profundidad $O(n^2)$.

La ejecución de la subrutina cuántica del algoritmo de Shor depende del valor de t, que a partir la ecuación (4.46) se elige según la precisión y exactitud deseadas. Vamos a considerar $t = O(n)$ a fines de este análisis. Se aplican t veces la secuencia de compuertas sobre el primer qubit y las compuertas $U_{a^{2^j}}$, por lo que los recursos computacionales necesarios son $2n+3$ qubits, $O(tn^2k_{max}) = O(n^3k_{max})$ compuertas y $O(tn^2) = O(n^3)$ de profundidad. Si tomamos un $k_{max} \propto \log(n/\epsilon)$, entonces el número de compuertas es $O(tn^2\log(n)) = O(n^3\log(n))$ para cualquier ϵ polinómico en $1/n$.

4.5. Implementación con 2n+2 qubits del Algoritmo de Shor

En esta sección se desarrollan algoritmos alternativos para la suma modular, que permiten prescindir de un qubit ancilla en la implementación del algoritmo de Shor, requiriendo entonces $2n+2$ qubits. Basándose en el trabajo de Y. Takahashi y N. Kunihiro, [2], y de T. Häner et al., [9], se ataca el problema de la adición modular de la sección 4.4.2 usando un circuito específico para comparar el estado del registro $|b\rangle$ con $N-a$, utilizando n qubits en vez de $n+1$. Para hacer esto,

es clave el uso de qubits llamados dirty ancilla, es decir qubits de trabajo sucios, donde por sucios se refiere a que no están inicializados a un estado particular conocidos. Estos qubits son usados como ayuda en los cómputos de la adición modular y al final de esta se encuentra en el mismo estado que estaban antes.

En la sección anterior vimos que en el algoritmo de Beauregard se usa la adición en el espacio de Fourier de Draper para comparar el estado del registro $|b\rangle$ con $N - a$, y así saber si $a + b$ es mayor o menor que N. Para hacer esto se necesita $n+1$ qubits, un qubit más que el tamaño de N, para almacenar la cifra más significativa que contiene la información que buscamos, pero luego, es necesario pasarla a otro qubit ancilla más, ya que se necesita seguir modificando el registro con sus $n+1$ qubits. Veremos ahora que es posible ahorrarse un qubit ancilla usando un circuito específico para obtener sólo el qubit mas significativo de la suma de $|b + a\rangle$, sin modificar $|b\rangle$.

En el circuito 4.16 se muestra la idea general del nuevo circuito para la adición modular

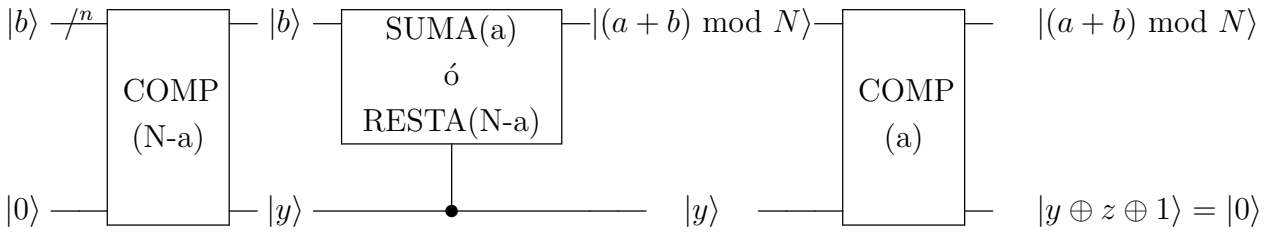


Figura 4.16: Circuito para la adición modular usando compuertas de comparación y un solo qubit ancilla. Vale que $y = 1 \iff b < N - a$ y $z = 1 \iff (a + b) \bmod N < a$.

Según y sea 0 o 1, se aplicará una suma por a o una resta por $N - a$ respectivamente.

4.5.1. Comparador

La pieza fundamental de este algoritmo es la compuerta COMP(a), que tiene como entrada un registro $|b\rangle$ de $n + 1$ qubits y devuelve el bit n de la suma, $|(b + a)_n\rangle$, dejando los n qubits restantes de $|b\rangle$ inalterados. Esta compuerta se basa en una compuerta de adición por propagación de acarreo con compuertas de TOFFOLI, donde se usan $n - 1$ qubits ancilla sucios para propagar información. Estos qubits de trabajo se encuentran en el primer registro del algoritmo de Shor, en el cuál cada vez que se ejecute una adición modular uno de los qubits tomará la función de ser uno de los qubits de control, mientras que los demás $n - 1$ qubits están disponibles para ser usados. Como no podemos saber el estado de los qubits ancilla sin medirlos, sólo pueden ser usados para propagar información mediante su inversión ([16]). Entonces, usamos estos qubits para propagar el bit de acarreo de cada cifra, es decir, la inversión del qubit $|g_i\rangle$, $g_i = 1$, indica que hay un acarreo desde la cifra i a la $i + 1$. Luego, el qubit $|g_i\rangle$ se invierte si se cumple al menos alguna de las

siguientes condiciones:

$$b_i = a_i = 1 \quad g_{i-1} = b_i \quad g_{i-1} = a_i = 1 \tag{4.96}$$

Sólo si el bit $c_i = 1$, entonces el qubit $|g_{i+1}\rangle$ se invierte si $b_i = 1$, lo cuál puede lograrse con una compuerta de CNOT sobre el qubit $|g_{i+1}\rangle$ y controlada por el qubit $|b_i\rangle$. Además, puede existir un acarreo cuando $b_i = 0$ pero $g_{i-1} = 1$, para lo cuál se aplica una compuerta de TOFFOLI sobre $|b_i\rangle$ y $|g_{i-1}\rangle$ después de una NOT en $|b_i\rangle$. Si $a_i = 0$, sólo puede generarse un acarreo al siguiente bit si $b_i = g_{i-1} = 1$, lo cuál se logra con la misma TOFFOLI anterior, sin aplicar la compuerta NOT.

Por lo tanto, para construir el circuito uno debe colocar compuertas de TOFFOLI sobre $|g_i\rangle$, controladas por $|g_{i-1}\rangle$ y $|b_i\rangle$, independientemente de a. Luego, sólo si $a_i = 1$, se agregan además una CNOT sobre $|g_i\rangle$ controlada por $|b_i\rangle$ y luego una NOT sobre $|b_i\rangle$. El orden de estas compuertas debe ser el siguiente: las compuertas TOFFOLI controladas en los qubits $|g_{i-1}\rangle$ son ejecutadas antes de que dicho qubit pueda ser invertido, y de nuevo una vez más, luego de que todas hayan sido ejecutada, para que de esta manera si ambas TOFFOLI controladas por el mismo $|g_{i-1}\rangle$ son ejecutadas, se cancelen entre sí. Una vez hayamos conseguido $|(a+b)_n\rangle$ en el último qubit, se revierte toda la secuencia de compuertas, excepto aquellas sobre este qubit, para devolver el registro $|g\rangle$ a su estado inicial.

Para i, notemos que no necesitamos $|g_0\rangle$ ya que no como no puede haber acarreo, podemos controlar la TOFFOLI directamente en $|b_0\rangle$ en vez de usarlo invertir $|g_0\rangle$. En este i tampoco debemos ejecutar dos de estas TOFFOLIs, ya que si $a_0 = 0$ no se ejecutan, mientras que si $a_1 = 1$ al ser ambas controladas por $|b_0\rangle$ se cancelarían.

Consideremos $|g\rangle = |g_{n-1}\rangle \otimes \cdots \otimes |g_1\rangle$ los $n-1$ qubits ancilla y $|b\rangle$ el registro al cual queremos sumarle un número clásico $a = \sum_{i=0}^{n} a_i 2^i$. Según la cifra binaria a_i sea 1 o 0, se ejecutará o no una serie de compuertas sobre los registros, sumando efectivamente dicha cifra. En 4.17 se muestra el circuito, donde con el fin de lograr mayor claridad ordenamos los qubits de manera que los que corresponden a la misma cifra binaria se encuentren juntos, a pesar de pertenecer a diferentes registros que luego en el algoritmo de Shor entero estarán separados.

Si observamos el circuito, podemos ver que si la entrada es un registro $|b\rangle$ de n qubits, es decir $b_n = 0$, entonces el circuito nos permite calcular el qubit más significante de la suma con a de $n+1$ bits *sin alterar* $|b\rangle$. Veamos un ejemplo, donde queremos obtener el qubit n de la suma de $a = 29$ al registro $|b\rangle$ de 4 qubits. Como las cifras binarias de a son $a_4a_3a_2a_1a_0 = 11101$, sólo se ejecutan las compuertas correspondientes a los bits $a_0,\ a_2,\ a_3$ y a_4. Entonces, el circuito entero es 4.18

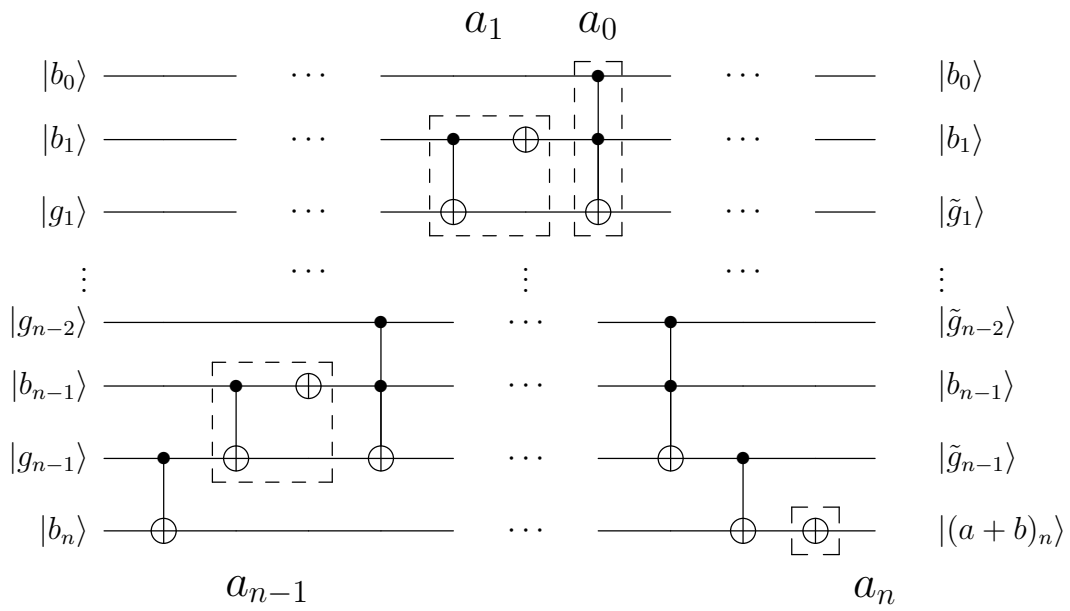


Figura 4.17: Circuito que calcula el bit más significativo de la suma del registro $|b\rangle$ de $n+1$ qubits y el número clásico a, utilizando $n-1$ qubits ancilla en el registro $|g\rangle$. Las compuertas encerradas por la linea de trazos sobre la cifra i son ejecutadas si y sólo si el bit a_i es 1. Una vez obtenido $|(a+b)_n\rangle$ se recorre la secuencia inversa de compuertas para revertir los qubits ancilla y los n qubits restantes de $|b\rangle$ a su estado inicial.

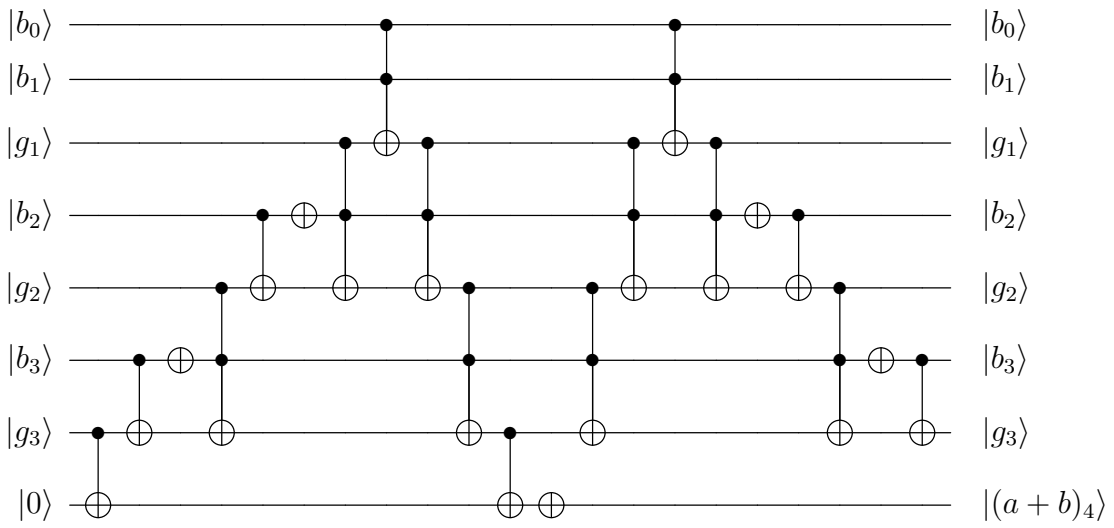


Figura 4.18: Circuito que calcula el bit más significativo de la suma del registro $|b\rangle$ de 4 qubits y el número $a = 29$.

4.5.2. Adición, Multiplicación y Exponenciación modular

Usando el comparador, ya podemos construir completamente el circuito cuántico para la adición modular, a partir de la idea del circuito 4.16. Esta implementación necesita entonces $2n+2$ qubits, uno de control, n en el primer registro, n en el segundo y un qubit auxiliar para la suma modular. En el 4.19 se muestra el circuito completo, teniendo en cuenta que cada compuerta COMP utiliza un qubit del primer registro como control y $n-1$ como ancillas. Notemos que como la compuerta COMP(a) da el último qubit de la suma de $|b\rangle$ y a, tenemos que hacer un pequeño cambio sobre ella para que efectivamente la salida del qubit ancilla sea $y = 1 \iff b < a$. Para ello, en vez de tomar a directamente vamos a usar su complemento de a dos de $n+1$ qubits $\bar{a} = 2^{n+1} - a$. Vemos que, como $a, b < 2^n$,

$$\begin{gathered} y = 1 \iff (b+2^n - a)_n = 1 \iff \\ (b + 2^{n+1} - a) \bmod 2^{n+1} \equiv (b-a) \bmod 2^{n+1} > 2^n \iff b < a \end{gathered} \tag{4.97}$$

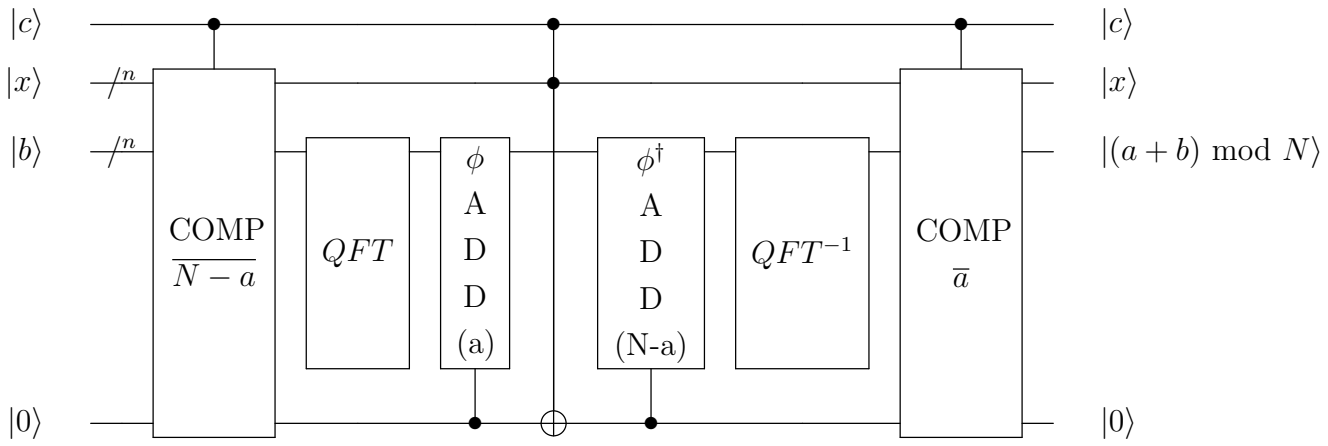


Figura 4.19: Circuito para la adición modular en la implementación de $2n+2$ qubits del algoritmo de Shor. En las compuertas COMP y en la compuerta de TOFFOLI uno de los qubits del registro $|x\rangle$ es usado junto con $|c\rangle$ como control, mientras que $n-1$ qubits de $|x\rangle$ son usados como qubits de trabajo en las compuertas COMP.

Los dos qubits de control de las compuertas COMP no se muestran en el circuito 4.17, pero podemos notar que como sólo el último qubits es alterado, sólo hay que controlar las compuertas que actúen sobre este. En particular, se reemplazan las compuertas CNOT y NOT sobre el ultimo qubit por compuertas C3NOT y TOFFOLI controladas por el qubit $|c\rangle$ y uno de los qubits del registro $|x\rangle$.

A partir de aquí, esta implementación es igual a la de la sección anterior, utilizando las compuertas recién construidas para implementar la multiplicación modular y las compuertas U_a. Tomando

como referencia el circuito 4.15, la única diferencia sería que en vez de tener un registro $|0\rangle$ con $n+2$ qubits ancilla, sólo se requieren $n+1$ qubits.

4.5.3. Análisis de Complejidad

De igual forma que como hicimos en la sección anterior, vamos ahora a estudiar la complejidad computacional de utilizar el circuito descrito en esta sección en la factorización de un número N de n bits. Vamos a considerar también que se pueden llevar a cabo simultáneamente compuertas sobre diferentes qubits, pero es imposible que un mismo qubit sirva como control en dos compuertas al mismo tiempo. Este circuito usa a lo más tres qubits de control en las compuertas C3NOT usadas para controlar la compuerta COMP, y como vimos en 3.2.3 estas se pueden implementar mediante una cantidad constante de compuertas controladas sobre un qubit.

La compuerta COMP requiere $2n$ qubits en total, contando el ancilla limpio y los $n-1$ ancillas sucios, precisa de $O(n)$ compuertas y tiene $O(n)$ profundidad algorítmica. Mientras tanto, las compuertas QFT y luego de suma y resta en el espacio de Fourier sólo requieren n ya que a diferencia con la implementación de la sección anterior, gracias al comparador podemos asegurar la suma o la resta esta siempre dentro del rango $[0, 2^n - 1]$. Las compuertas QFT y $\phi\ ADD$ tienen $O(nk_{max})$ compuertas y (n) profundidad. Estas compuertas que se usan aquí sólo tiene un qubit de control, lo que a diferencia de la implementación anterior donde la mayor parte de las $\phi\ ADD$ tienen dos controles, permiten reducir el número de compuertas en una cantidad constante en n, lo cuál puede hacer la diferencia para N pequeño. La adición modular requiere entonces $2n+2$ qubits, teniendo en el uso de los qubit ancilla sucios. Como estos qubits están disponibles en el primer registro, resulta que desde las compuertas fundamentales se usan todos los qubits de la implementación. El análisis de la complejidad es idéntico a partir de aquí al de la sección anterior, ya que aquí también podemos hacer uso de la QFT aproximada y tanto la multiplicación modular como la exponenciación son implementadas de manera equivalente, sólo cambiando la compuerta de adición modular. Por lo tanto, para cierto $t = O(n)$, este circuito para implementar el algoritmo de Shor requiere $2n+2$ qubits, $O(tn^2 \log n) = O(n^3 \log n)$ compuertas y $O(tn^2) = O(n^3)$ profundidad, si se elige para la QFT $k_{max} = O(\log(n/\epsilon))$.

Capítulo 5

Implementación Numérica, Experimentos y Resultados

En los capítulos 3 y 4 se presentaron las bases teóricas tanto de vectores de estado y compuertas fundamentales como de algoritmos, y las técnicas algebraicas para maximizar la eficiencia de una posterior implementación numérica.
En este capítulo, se aborda el problema de la simulación de estas compuertas y algoritmos, presentando primero en la sección 5.1 los desafíos y soluciones propios del paso de la teoría al código. Luego, en la sección 5.2 se adaptan las expresiones algebraicas de las compuertas fundamentales de la sección 3.2 para su implementación en varios procesos paralelos.

5.1. Implementación Numérica

5.1.1. Implementación en FORTRAN

En este trabajo, se utilizará el lenguaje de programación compilado *FORTRAN*. El mismo es un lenguaje de alto nivel, desarrollado por la empresa IBM en la década de los 50's para ser usado en sus máquinas IBM 704. Está altamente optimizado para el cálculo numérico y la computación científica, sobre todo a la hora de tratar con arreglos de gran tamaño y gran cantidad de operaciones aritméticas, que es precisamente de lo que trata este trabajo. Es uno de los lenguajes más populares en el área de la computación de alto rendimiento y es el lenguaje usado para programas que evalúan el desempeño (benchmark) y el ranking de los supercomputadores más rápidos del mundo. Citando su propia pagina, algunas de sus ventajas incluyen:

- Alto desempeño
- Simple de aprender y usar

- Versatilidad
- Fácilmente paralelizable
- Facilidad del compilador para encontrar errores y advertencias en el código

Se utilizara el módulo `mzranmod` el generador de números pseudo-aleatorios de Marsaglia & Zaman, obtenido del libro *Numerical Recipes* [20].

5.1.2. Aspectos generales de la Implementación Numérica

En base al marco teórico desarrollado en la sección 3, se simulará la evolución de vectores de estado de varios qubits al aplicarse sobre ellos distintas compuertas cuánticas. Como vimos, un vector de estado general de n qubits puede escribirse como

$$|\psi\rangle = \sum_{x=0}^{2^n-1} a_x |x\rangle \tag{5.1}$$

En general, estos vectores de estado están entrelazados, es decir, no pueden escribirse como el producto tensorial de los vectores de estado de n qubits individuales:

$$|\psi\rangle \neq (a_{0,0}|0\rangle + a_{0,1}|1\rangle) \otimes \cdots \otimes (a_{n-1,0}|0\rangle + a_{n-1,1}|1\rangle) \tag{5.2}$$

A pesar de iniciar la simulación vectores con no entrelazados, las compuertas controladas como $CNOT$ y $CR(\phi)$ producen entrelazamiento. Veamos un ejemplo: apliquemos una compuerta $CNOT$ a dos qubits, uno inicialmente en el estado $|+\rangle = (|0\rangle + |1\rangle)/\sqrt{2}$ y el otro en el estado $|0\rangle$. Entonces, el estado inicial es

$$|\psi_i\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |2\rangle) \tag{5.3}$$

La acción de $CNOT$ es invertir el segundo qubit si el primero se encuentra en el estado excitado, osea

$$CNOT|\mathbf{0}\rangle = |\mathbf{0}\rangle \quad CNOT|\mathbf{1}\rangle = |\mathbf{1}\rangle \quad CNOT|\mathbf{2}\rangle = |\mathbf{3}\rangle \quad CNOT|\mathbf{3}\rangle = |\mathbf{2}\rangle \tag{5.4}$$

Por lo tanto

$$|\psi_f\rangle = CNOT|\psi_i\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle) = \frac{1}{\sqrt{2}}(|\mathbf{0}\rangle + |\mathbf{3}\rangle) = |\Phi^+\rangle \tag{5.5}$$

donde $|\Phi^+\rangle$ es el primer estado de Bell.

Debido a este entrelazamiento se debe almacenar siempre los 2^n coeficientes del vector producto tensorial completo y aplicar sobre este las correspondientes compuertas. Es este incremento exponencial en el tamaño del vector de estado con el número de qubits lo que hace estas simulaciones en computadoras clásicas muy costosas computacionalmente. Más aún, los operadores están representados por matrices $2^n \times 2^n$, por lo que resulta clara la ya mencionada impracticabilidad de aplicar los operadores simplemente multiplicando matrices por vectores.

5.1.3. Inicialización

Al iniciar una simulación y antes de empezar a aplicar las compuertas, se desea o bien tener un estado con entrelazamiento arbitrario, o bien un estado libre de entrelazamiento. Esta distinción resulta fundamental a la hora de escoger el algoritmo de inicialización. El primer caso es mas sencillo, ya que la única condición a cumplir por los coeficientes es que $1 = \sum_{x=0}^{2^n-1} |a_x|^2$. Entonces, tanto si se quiere un vector predefinido o un vector completamente aleatorio, para inicializar un vector de estado de n qubits solo hay que asignar un valor a cada uno de los 2^n coeficientes y luego normalizar el vector dividiéndolo por su propio módulo.

Por otro lado, si se quiere un vector libre de entrelazamiento, deben tenerse mas cosas en cuenta. Primero debe inicializarse el vector de estado individual de cada qubit para luego realizar el producto tensorial de todos ellos para obtener el vector de estado de todo el sistema. Como se explica en la sección 3.1.2, es posible representar inequívocamente todos los estados posibles de un qubit mediante solo dos coeficientes reales.

$$|\psi\rangle = cos\Big(\frac{\theta}{2}\Big)\,|0\rangle + e^{i\varphi} sin\Big(\frac{\theta}{2}\Big)\,|1\rangle \tag{5.6}$$

Los coeficientes φ y θ definen inequívocamente un punto en la Esfera de Bloch. Por lo tanto, a la hora de inicializar cada qubit, solo hay que elegir φ y θ. Por ejemplo, si quiero inicializar el qubit en el estado fundamental, simplemente elijo $\theta = 0$ y $\varphi = 0$. Si, por otro lado, quiero tener un vector aleatorio con distribución uniforme sobre la esfera de Bloch, sorteo $\theta \in (0, \pi)$ y $\varphi \in (0, 2\pi)$.

El siguiente paso es implementar el producto tensorial, lo cuál es computacionalmente costoso. Empecemos por el caso en que $n = 2$ y sean $|\psi_0\rangle = a_{00}\,|0\rangle + a_{01}\,|1\rangle$ y $|\psi_1\rangle = a_{10}\,|0\rangle + a_{11}\,|1\rangle$. Entonces, el vector producto tensorial es

$$|\psi\rangle = a_{00}a_{10}\,|0\rangle + a_{00}a_{11}\,|1\rangle + a_{01}a_{10}\,|2\rangle + a_{01}a_{11}\,|3\rangle \tag{5.7}$$

Puede observarse que a_{10} es factor de los coeficientes con número par, mientras que a_{11} de los impares. Por su parte, a_{00} es factor de los coeficientes 0 y 1.

Si se observa con atención puede verse que a_{00} es factor del coeficiente número k tal que

$$\frac{k}{2^{n-1}} \equiv \frac{k}{2^1} \equiv \frac{k}{2} \equiv 0 \mod 2 \quad \Longrightarrow\ k = 0, 1 \tag{5.8}$$

a_{01} es factor del coeficiente número k tal que

$$\frac{k}{2^{n-1}} \equiv \frac{k}{2^1} \equiv \frac{k}{2} \equiv 1 \mod 2 \quad \Longrightarrow\ k = 2, 3 \tag{5.9}$$

a_{10} es factor del coeficiente número k tal que

$$\frac{k}{2^{n-1-1}} \equiv k \equiv 0 \mod 2 \quad \Longrightarrow\ k = 0, 2 \tag{5.10}$$

mientras que a_{11} es factor del coeficiente número k tal que

$$\frac{k}{2^{n-1-1}} \equiv k \equiv 1 \mod 2 \quad \implies k = 1, 3 \tag{5.11}$$

Luego, sea $i = 0, 1$ el número del qubit y $j = 0, 1$ si corresponde al estado $|0\rangle$ o $|1\rangle$ respectivamente, entonces a_{ij} es factor de aquellos coeficientes número k tales que

$$\frac{k}{2^{n-(i+1)}} \equiv j \mod 2 \tag{5.12}$$

En el caso general en que $i = 0, \ldots, n-1$ el coeficiente a_k con $k \in (0, 2^n - 1)$ puede construirse mediante la siguiente productoria

$$a_k = \prod_{i=0}^{n-1} a_{i,\mathrm{mod}(k/2^{n-(i+1)},2)} \tag{5.13}$$

5.1.4. Aplicación de Compuertas Cuánticas

La forma mas directa de aplicar operadores cuánticos al vector producto tensorial es simplemente multiplicar la matriz $2^n \times 2^n$ correspondiente al operador por el vector de estado. Sin embargo, como puede verse en los desarrollos de la sección 3.2, dichas matrices son a lo sumo tridiagonales, por lo que si se hace directamente la multiplicación de matrices se llevarían a cabo muchas multiplicaciones por cero, ya que la mayor parte de los 2^{2n} coeficientes de las matrices son cero. También, como para extender operadores de uno o dos qubits a todo el sistema de n qubits se hace el producto tensorial por identidades, se tiene que gran parte de los coeficientes no nulos son irrelevantes, ya que no cambian el vector de estado. La notación producto externo en la cuál se expresan todos los operadores desarrollados en la sección 3.2 facilita una implementación mas eficientes de las compuertas cuánticas. Retomando el ejemplo (5.5), la acción de aplicar la compuerta CNOT a un vector de estado de 2 qubits $|\psi\rangle$ puede ser expresada, en notaciones matricial y producto externo, de la siguiente manera

$$\begin{aligned} CNOT\,|\psi\rangle &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} a_0 \\ a_1 \\ a_3 \\ a_2 \end{bmatrix} \\ &= (|0\rangle\langle 0| + |1\rangle\langle 1| + |3\rangle\langle 2| + |2\rangle\langle 3|)\,|\psi\rangle \\ &= (|0\rangle\langle 0| + |1\rangle\langle 1| + |3\rangle\langle 2| + |2\rangle\langle 3|)(a_0\,|0\rangle + a_1\,|1\rangle + a_2\,|2\rangle + a_3\,|3\rangle) \\ &= a_0\,|0\rangle + a_1\,|1\rangle + a_3\,|2\rangle + a_2\,|3\rangle \end{aligned} \tag{5.14}$$

Como $(b_{ki}\,|k\rangle\langle i|)\;a_j\,|j\rangle = a_j b_k \delta_{ij}\,|k\rangle$, la interpretación es que el *ket* indica cuál coeficiente del vector de estado va a ser reemplazado por el valor del coeficiente del *bra* multiplicado por

el elemento de matriz b_{ki} del operador. En el caso de haber varios *bra*, el coeficiente del *ket* se reemplaza por la suma de los valores de los *bra* por los correspondientes elementos de matriz.

Entonces, para aplicar una compuerta, simplemente hay que reemplazar el coeficiente de cada *ket* que aparezca en la expresión producto externo del operador, por la suma de los coeficientes por los elementos de matriz, según corresponda. Es importante notar que, como se dijo antes, hay coeficientes que a pesar de no ser nulos, no cambian el vector de estado. Por ejemplo, cuando la suma de los coeficientes por los elementos de matriz sólo tiene el elemento de matriz sobre la diagonal igual a 1, esto se corresponde a reemplazar el coeficiente del vector por si mismo, por lo que esta operación puede ignorarse.

De esta forma, se reduce enormemente el número de operaciones a realizar al momento de aplicar un operador, comparado con la multiplicación de matrices. Para cada una de las compuertas básicas, el número de sumas y multiplicaciones de números complejos escala en función del tamaño del sistema con $2^n 3$ en el caso de la compuerta de Hadamard, 2^{n-1} en el caso de CNOT, 2^{n-1} en el caso de R(ϕ) y 2^{n-2} para CR(ϕ).

5.2. Paralelización

Si bien cada compuerta actúa sobre cierto (o ciertos) qubit, debido al entrelazamiento a la hora de las simulaciones numéricas es necesario modificar directamente el vector de estado de todo el sistema, con sus 2^n coeficientes complejos. Retomando las expresiones en la forma de producto externo desarrolladas en la sección 3.2 para cada compuerta, se desarrollaran en esta sección adaptaciones para permitir su implementación en muchos procesos paralelos. Dicha implementación se logra en FORTRAN usando la interfaz de programación paralela OpenMP[21].

La idea fundamental es la siguiente. Sin producir copias, cada proceso toma una fracción del vector de estado del sistema, que se determina partiendo alguna de las sumatorias del producto externo. Luego, como vimos que las compuertas son a lo más tridiagonales pero cada coeficiente puede ser mezclado a lo mucho consigo mismo y sólo un coeficiente más, es decir $q(x) \to aq(x) + bq(x \pm c)$, cada proceso puede usar y modificar la fracción del vector de estado que se le asigno, sin necesitar ni tener que prestar nada a los demás procesos. Al terminar todos los procesos, el vector de estado ha sido modificado en su totalidad, parte por parte, igual que si se hubiera hecho de manera secuencial.

Como es de esperar surgen muchas complicaciones y limitaciones, pero la mayoría pueden solventarse programando casos particulares según sean n, l y m, el número de qubits total, el índice del qubit sobre el que actúa la compuerta y el índice del qubit de control, respectivamente. Como los límites de las sumatorias son potencias de dos, con el fin de asegurarse que al partir las mismas en los distintos procesos no se producen sino divisiones enteras, es necesario que el número

de procesos paralelos también sea potencia de 2. Aún así, como se comprobará en la sección siguiente 5.3, para n pequeños y una cantidad grande de procesos paralelos irremediablemente se presentarán problemas, cuando el número de procesos sea mayor a los índices de la sumatoria. Esto no resultará un gran problema, porque justamente la utilidad de la paralelización es simular sistemas grandes.

A continuación, se realizará la adaptación de las expresiones algebraicas de las compuertas fundamentales. No es necesario ver cada compuerta fundamental de un qubit; por el contrario se analizaran compuertas generales de un qubit y compuertas de un qubit con un control. En los programas, las compuertas con mayor uso como la CNOT, la Hadamard o la CR(ϕ) se implementan adaptando las expresiones algebraicas que se mostrarán en las siguientes ecuaciones, mientras que las menos comunes, como pueden ser las necesarias para implementar la compuerta de TOFFOLI o la C3NOT (3.2.3), se implementan llamando a la compuerta controlada general de un qubit, pero especificando a, b, c y d.

5.2.1. Paralelización de Compuertas de un qubit

Consideremos un operador U sobre un qubit dado por

$$U = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \quad \Longrightarrow \quad U = a\,|0\rangle\langle 0| + b\,|0\rangle\langle 1| + c\,|1\rangle\langle 0| + d\,|1\rangle\langle 1| \tag{5.15}$$

Como se deduce en la subsección 3.2.1, la forma producto externo de la aplicación de U sobre el qubit l-ésimo esta dada por la ecuación (3.62):

$$\begin{aligned} I_A \otimes U \otimes I_B = \sum_{k=0}^{A-1}\sum_{i=0}^{B-1}(&|2kB+i\rangle\,(a(\langle 2kB+i| + b\,\langle (2k+1)B+i|) \\ &+ |(2k+1)B+i\rangle\,((c(\langle 2kB+i| + d\,\langle (2k+1)B+i|)) \end{aligned} \tag{5.16}$$

donde $A = 2^{l-1}$ y $B = B$.
Con el objetivo de reducir la ocurrencia de divisiones no enteras, se consideran los casos particulares en que $B \geq A$, o sea, $z - l \geq l - 1$, y el caso contrario. Se destacarán los cambios que presenta cada caso con respecto a la versión no paralelizada en negrita dentro del *ket* o *bra* correspondiente

Compuertas de un qubit: $z - l \geq l - 1$

Como $B \geq 2^{l-1}$, entonces la sumatoria sobre k puede partirse entre más procesos paralelos, precisamente en $p = B$ procesos. Entonces, la expresión producto externo es

$$\begin{aligned} I_A \otimes U \otimes I_B = & \sum_{r=0}^{p-1} \sum_{k=0}^{A-1} \sum_{i=0}^{B/p-1} \\ & (|2kB + \mathbf{rB/p} + i\rangle \, (a \, \langle 2kB + \mathbf{rB/p} + i| + b \, \langle (2k+1)B + \mathbf{rB/p} + i|) \\ & + |(2k+1)B + \mathbf{rB/p} + i\rangle \, (c \, \langle 2kB + \mathbf{rB/p} + i| + d \, \langle (2k+1)B + \mathbf{rB/p} + i|)) \end{aligned} \tag{5.17}$$

Compuertas de un qubit: $z - l < l - 1$

Por su parte, cuando $B < A$, pueden usarse hasta $p = 2^{l-1}$

$$\begin{aligned} I_A \otimes U \otimes I_B = & \sum_{r=0}^{p-1} \sum_{k=0}^{A/p-1} \sum_{i=0}^{B-1} \\ & (|2kB + \mathbf{2rAB/p} + i\rangle \, (a \, \langle 2kB + \mathbf{2rAB/p} + i| + b \, \langle (2k+1)B + \mathbf{2rAB/p} + i|) \\ & + |(2k+1)B + \mathbf{2rAB/p} + i\rangle \, (c \, \langle 2kB + \mathbf{2rAB/p} + i| + d \, \langle (2k+1)B + \mathbf{2rAB/p} + i|)) \end{aligned} \tag{5.18}$$

5.2.2. Paralelización de Compuertas de un qubit controladas

Sólo vamos a deducir como paralelizar las compuertas con un qubit de control y uno de acción, como CNOT y CR(ϕ), ya que estas son las fundamentales. Generalizando los casos particulares de las ecuaciones (3.87)(3.86)(3.88), cuando se aplica un operador U sobre el qubit l y controlado por el qubit m-ésimo, se distinguen dos casos particulares. Definiendo en cada caso A, B y C tenemos:

$$l > m \implies A = 2^{l-m+1} \quad B = 2^{z-l} \quad C = 2^{m-1}$$

$$\begin{aligned} I_C \otimes U_{ml} \otimes I_B = & \sum_{k=0}^{C-1} \sum_{j=0}^{A/4-1} \sum_{i=0}^{B-1} \\ & \Big(|(2j + A/2)B + kAB + i\rangle \, (a \, \langle (2j + A/2)B + kAB + i| + b \, \langle (2j + A/2 + 1)B + kAB + i|) \\ & + |(2j + A/2 + 1)B + kAB + i\rangle \, (c \, \langle (2j + A/2)B + kAB + i| + d \, \langle (2j + A/2 + 1)B + kAB + i|) \Big) \end{aligned} \tag{5.19}$$

$$l < m \implies A = 2^{z-l} \quad B = 2^{z-m} \quad C = 2^{l-1}$$

$$\begin{aligned} I_C \otimes U_{ml} \otimes I_B = & \sum_{k=0}^{C-1} \sum_{j=0}^{AB/2-1} \sum_{i=0}^{B-1} \\ & \Big(|2kA + (2j+1)B + i\rangle \left(a \langle 2kA + (2j+1)B + i| + b \langle (2k+1)A + (2j+1)B + i|\right) \\ & + |(2k+1)A + (2j+1)B + i\rangle \left(c \langle 2kA + (2j+1)B + i| + d \langle (2k+1)A + (2j+1)B + i|\right) \Big) \end{aligned} \tag{5.20}$$

En las expresiones de arriba, siguiendo lo dicho en la subsección 5.1.4, se explicitan sólo los elementos de matriz que realmente cambian el coeficiente del vector de estado. Es decir, no se muestra ningún elementos del tipo $|i\rangle \langle i|$ ya que estás operaciones redundantes no se realizan cuando se aplican las compuertas.

A la hora de adaptar las ecuaciones (5.19),(5.20), surgen casos particulares según cuál A, B o C es el mayor, además de si l es mayor que m, para un total de seis casos distintos. Para destacar los cambios que presenta cada caso con respecto a la versión no paralelizada, se los destacará en negrita dentro del *ket* o *bra* correspondiente

Compuertas controladas: $l > m$ y $l - m - 1 \geq z - l, m - 1$.

Si $l > m$ entonces $A = 2^{l-m+1}$, $B = 2^{z-l}$y $C = 2^{m-1}$. Si además $A/4 \geq B, C$, o sea, $l - m - 1 \geq z - l, m - 1$ se parte la sumatoria sobre el índice j en la ecuación (5.19), pudiéndose usar hasta $p = 2^{l-m-1}$ procesos paralelos.

$$\begin{aligned} I_C \otimes U_{ml} \otimes I_B = & \sum_{r=0}^{p-1} \sum_{k=0}^{C-1} \sum_{j=0}^{A/4p-1} \sum_{i=0}^{B-1} \\ & \Big(|(2j + A/2)B + kAB + \mathbf{rA/2p} + i\rangle \left(a \langle (2j + A/2)B + kAB + \mathbf{rA/2p} + i| \right. \\ & \left. + b \langle (2j + A/2 + 1)B + kAB + \mathbf{rA/2p} + i|\right) \\ & + |(2j + A/2 + 1)B + kAB + \mathbf{rA/2p} + i\rangle \left(c \langle (2j + A/2)B + kAB + \mathbf{rA/2p} + i| \right. \\ & \left. + d \langle (2j + A/2 + 1)B + kAB + \mathbf{rA/2p} + i|\right) \Big) \end{aligned} \tag{5.21}$$

Compuertas controladas: $l > m$ **y** $m-1 \geq z-l, l-m-1$**.**

Si $C \geq B, A$, o sea, $m-1 \geq z-l, l-m-1$ se parte la sumatoria sobre el índice k en la ecuación (5.19), pudiéndose usar hasta $p = 2^{m-1}$ procesos paralelos.

$$
\begin{aligned}
&I_C \otimes U_{ml} \otimes I_B = \sum_{r=0}^{p-1} \sum_{k=0}^{C/p-1} \sum_{j=0}^{A/4-1} \sum_{i=0}^{B-1} \\
&\Big(|(2j + A/2)B + kAB + \mathbf{rAB/2p} + i\rangle \, (a \, \langle (2j + A/2)B + kAB + \mathbf{rAB/2p} + i| \\
&+ b \, \langle (2j + A/2 + 1)B + kAB + \mathbf{rAB/2p} + i|) \\
&+ |(2j + A/2 + 1)B + kAB + \mathbf{rAB/2p} + i\rangle \, (c \, \langle (2j + A/2)B + kAB + \mathbf{rAB/2p} + i| \\
&+ d \, \langle (2j + A/2 + 1)B + kAB + \mathbf{rAB/2p} + i|) \Big)
\end{aligned}
\tag{5.22}
$$

Compuertas controladas: $l > m$ **y** $z-l \geq m-1, l-m-1$**.**

Si $B \geq C, A$, o sea, $z-l \geq m-1, l-m-1$ se parte la sumatoria sobre el índice i en la ecuación (5.19), pudiéndose usar hasta $p = 2^{z-l}$ procesos paralelos.

$$
\begin{aligned}
&I_C \otimes U_{ml} \otimes I_B = \sum_{r=0}^{p-1} \sum_{k=0}^{C/p-1} \sum_{j=0}^{A/4-1} \sum_{i=0}^{B-1} \\
&\Big(|(2j + A/2)B + kAB + \mathbf{rABC/2p} + i\rangle \, (a \, \langle (2j + A/2)B + kAB + \mathbf{rABC/2p} + i| \\
&+ b \, \langle (2j + A/2 + 1)B + kAB + \mathbf{rABC/2p} + i|) \\
&+ |(2j + A/2 + 1)B + kAB + \mathbf{rABC/2p} + i\rangle \, (c \, \langle (2j + A/2)B + kAB + \mathbf{rABC/2p} + i| \\
&+ d \, \langle (2j + A/2 + 1)B + kAB + \mathbf{rABC/2p} + i|) \Big)
\end{aligned}
\tag{5.23}
$$

Compuertas controladas: $l < m$ **y** $m-l \geq z-m, l-1$**.**

Si $l < m$ entonces $A = 2^{z-l}$, $B = 2^{z-m}$y $C = 2^{l-1}$. Si además $A/2B \geq B, C$, o sea, $m-l \geq z-m, l-1$ se parte la sumatoria sobre el índice j en la ecuación (5.19), pudiéndose usar hasta $p = 2^{m-l}$ procesos paralelos.

$$
\begin{aligned}
&I_C \otimes U_{ml} \otimes I_B = \sum_{r=0}^{p-1} \sum_{k=0}^{C-1} \sum_{j=0}^{A/2Bp-1} \sum_{i=0}^{B-1} \\
&\Big(|2kA + (2j + 1)B + \mathbf{rA/p} + i\rangle \, (a \, \langle 2kA + (2j + 1)B + \mathbf{rA/p} + i| \\
&+ b \, \langle (2k + 1)A + (2j + 1)B + \mathbf{rA/p} + i|) \\
&+ |(2k + 1)A + (2j + 1)B + \mathbf{rA/p} + i\rangle \, (c \, \langle 2kA + (2j + 1)B + \mathbf{rA/p} + i| \\
&+ d \, \langle (2k + 1)A + (2j + 1)B + \mathbf{rA/p} + i|) \Big)
\end{aligned}
\tag{5.24}
$$

Compuertas controladas: $l < m$ **y** $l - 1 \geq z - m, m - l$**.**

Si $C \geq B, A/2B$, o sea, $l - 1 \geq z - m, m - l$ se parte la sumatoria sobre el índice k en la ecuación (5.19), pudiéndose usar hasta $p = 2^{l-1}$ procesos paralelos.

$$
\begin{aligned}
I_C \otimes U_{ml} \otimes I_B = & \sum_{r=0}^{p-1} \sum_{k=0}^{C/p-1} \sum_{j=0}^{A/2B-1} \sum_{i=0}^{B-1} \\
& \Big(|2kA + (2j+1)B + \mathbf{2rAC/p} + i\rangle \left(a \langle 2kA + (2j+1)B + \mathbf{2rAC/p} + i| \right. \\
& + b \langle (2k+1)A + (2j+1)B + \mathbf{2rAC/p} + i| \big) \\
& + |(2k+1)A + (2j+1)B + \mathbf{2rAC/p} + i\rangle \left(c \langle 2kA + (2j+1)B + \mathbf{2rAC/p} + i| \right. \\
& + d \langle (2k+1)A + (2j+1)B + \mathbf{2rAC/p} + i| \big) \Big)
\end{aligned}
\tag{5.25}
$$

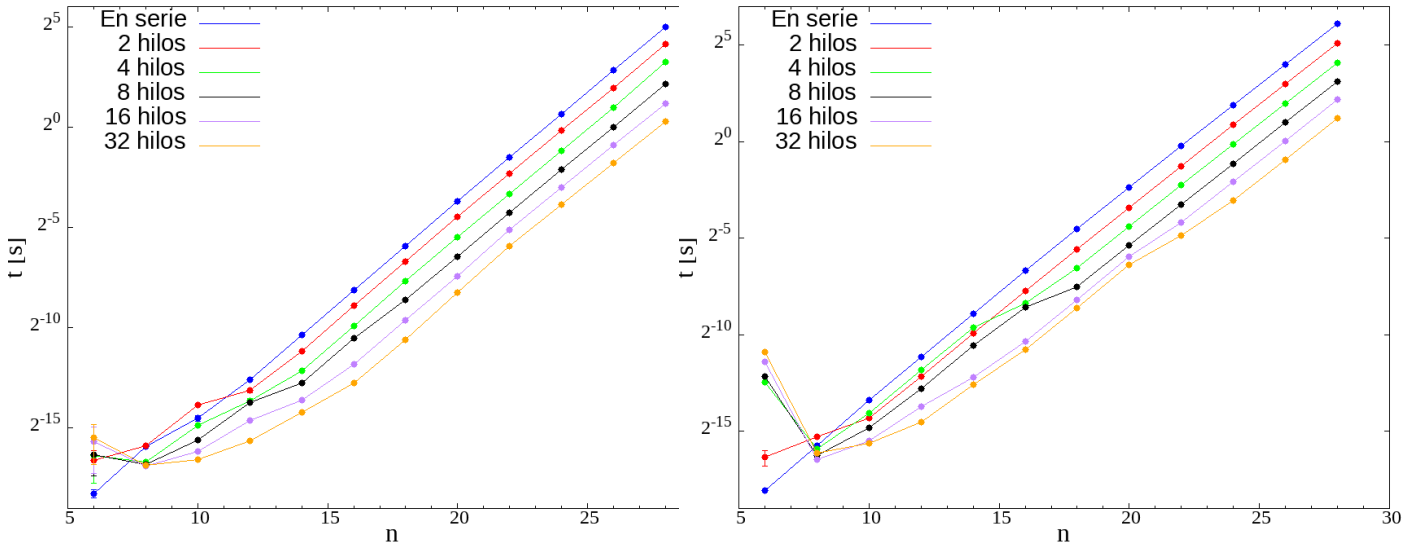
Compuertas controladas: $l < m$ **y** $z - m \geq m - l, l - 1$**.**

Si $B \geq C, A/2B$, o sea, $z - m \geq m - l, l - 1$ se parte la sumatoria sobre el índice i en la ecuación (5.19), pudiéndose usar hasta $p = 2^{z-m}$ procesos paralelos.

$$
\begin{aligned}
I_C \otimes U_{ml} \otimes I_B = & \sum_{r=0}^{p-1} \sum_{k=0}^{C/p-1} \sum_{j=0}^{A/2B-1} \sum_{i=0}^{B-1} \\
& \Big(|2kA + (2j+1)B + \mathbf{rB/p} + i\rangle \left(a \langle 2kA + (2j+1)B + \mathbf{rB/p} + i| \right. \\
& + b \langle (2k+1)A + (2j+1)B + \mathbf{rB/p} + i| \big) \\
& + |(2k+1)A + (2j+1)B + \mathbf{rB/p} + i\rangle \left(c \langle 2kA + (2j+1)B + \mathbf{rB/p} + i| \right. \\
& + d \langle (2k+1)A + (2j+1)B + \mathbf{rB/p} + i| \big) \Big)
\end{aligned}
\tag{5.26}
$$

5.3. Resultados y análisis de datos

En esta sección se mostrarán los resultados de las simulaciones descritas en la sección anterior. Se caracterizarán las mismas según su tiempo de ejecución, número de compuertas aplicadas, error numérico y algorítmico, analizando distintas opciones de paralelización. En cada caso se hace estadística con 30 experimentos, para de esta forma obtener de forma preliminar la media y la desviación estándar. Esta última se muestra como barras de error en los gráficos, y en los casos en que no se observe es debido a que la misma es despreciable. Para caracterizar el error de el vector calculado por las compuertas en relación al esperado teórica, se calcula (a menos que se especifique lo contrario) la proyección del vector calculado sobre el vector ideal y se le resta 1, correspondiente a una precisión absoluta. Luego, se calcula la media y la desviación estándar de esta diferencia.



(a) Simple precisión.

(b) Doble precisión.

Figura 5.1: Tiempo de ejecución de la subrutina de inicialización en función del número de qubits y la cantidad de hilos paralelos.

5.3.1. Inicialización

En los gráficos de la figura 5.1 se muestra el tiempo de ejecución de la operación descrita en la sección 5.1.3 de inicializar el vector de estado de todo el sistema como el producto tensorial de los vectores de estado de cada qubit. Se estudio si usar números de precisión o doble, lo cuál repercute en el tiempo de ejecución, como así también la paralelización, usando 2, 4, 8, 16 y 32 hilos.

Se puede observar como según lo esperado, el tiempo de ejecución escala asintóticamente como 2^n al igual que los coeficientes del vector de estado del sistema. También se observa como para n pequeños la paralelización no otorga mejoras significativas al tiempo de ejecución. Los gráfico para cada tipo de precisión tienen comportamiento muy similar, simplemente la precisión doble tarda más.

A partir de aquí y por simplicidad y conveniencia, sólo se tendrá en cuenta la precisión simple, ya que con esta es suficiente para los tamaños de sistemas con los que se va a tratar.

5.3.2. Compuertas Fundamentales

Hadamard

A continuación se estudiara el desempeño de las compuertas fundamentales, tanto en tiempo como en precisión. La primera que analizaremos es la compuerta de Hadamard. En las figuras 5.2 y 5.3 se muestra el tiempo de ejecución y el error respectivamente. Para calcular este error, se

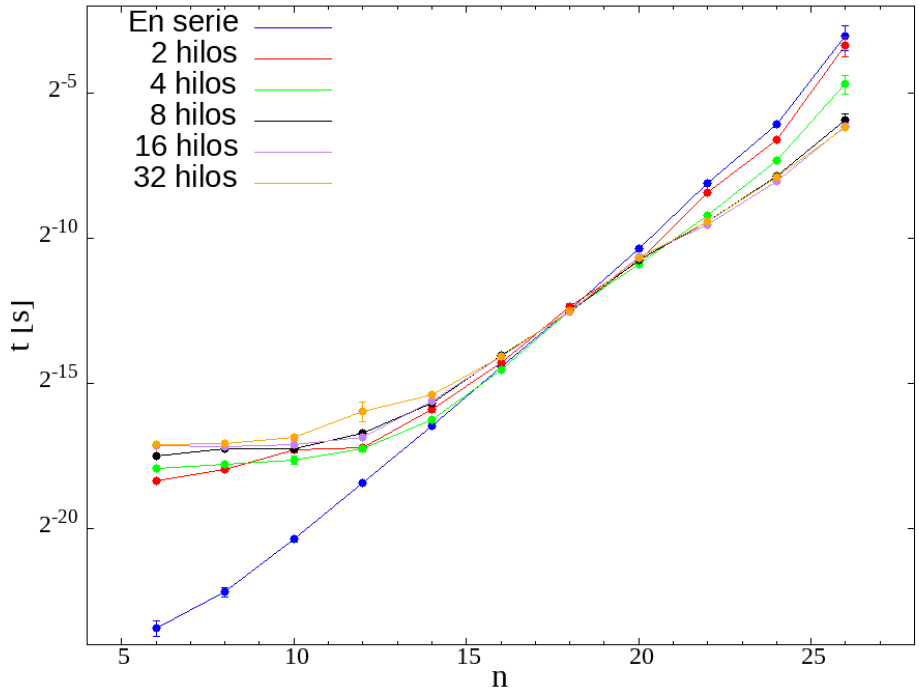


Figura 5.2: Tiempo de ejecución de la compuerta de Hadamard en función del número de qubits del sistema y la cantidad de hilos paralelos.

inicializa cada qubit al azar en el estado $|0\rangle$ o $|1\rangle$ y se calcula el vector de todo el sistema con el producto tensorial de los qubits. Luego, se le aplica la compuerta de Hadamard a un qubit l cualquier y finalmente, se compara el resultado de esta operación con el vector de estado del sistema que resulta de cambiar el qubit l al estado $|+\rangle$ o $|-\rangle$ según si inicialmente estaba en $|0\rangle$ o $|1\rangle$ respectivamente. Se observa en la figura 5.2 como la paralelización empieza a ser provechosa a partir de $n \sim 18$, debido a que comparando con la figura 5.1, el tiempo de ejecución es mucho menor y por lo tanto el costo fijo en tiempo que resulta de iniciar los procesos paralelos es más prevalente. Puede verse en 5.3 que, como se había avisado en la sección 5.2, las implementaciones paralelizadas presentan problemas para n pequeños. Estos serían los errores algorítmicos, mientras que para $n > 10$ el error de todas las opciones de paralelización se estabiliza, correspondiendo al error numérico de las operaciones de punto flotante. Este problema se observa en todas las compuertas paralelizadas, en mayor o menor medida.

$\mathbf{R}(\phi)$

La siguiente compuerta fundamental que analizaremos es la compuerta de rotación R(ϕ). En las figuras 5.4 y 5.5 se muestra el tiempo de ejecución y el error. Se observa un comportamiento similar en cuanto tiempo, pero se ve una mayor variación en el error. Esto puede deberse a que

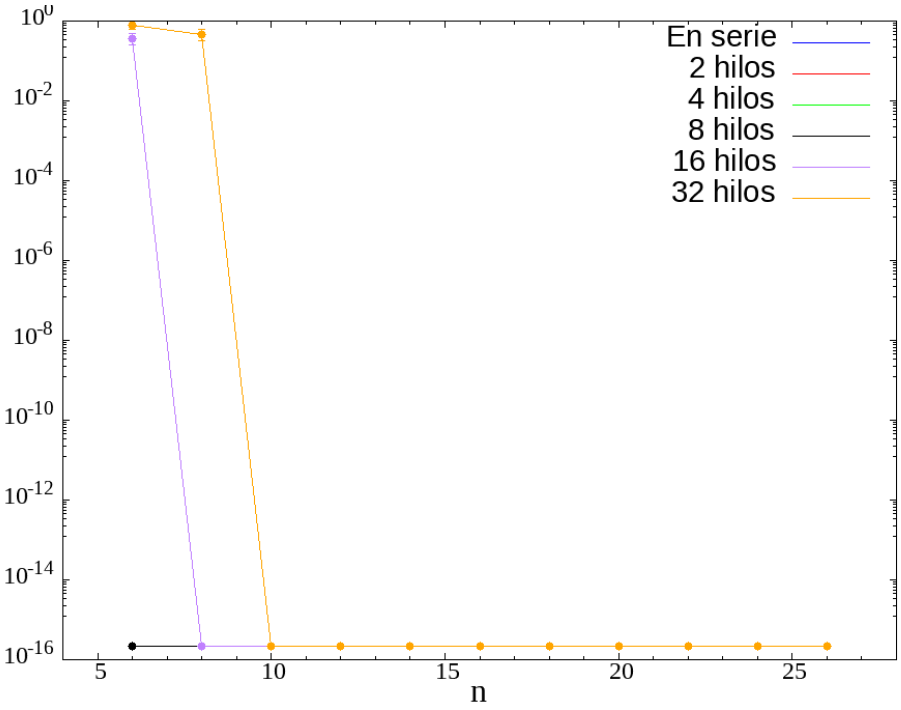


Figura 5.3: Error de la compuerta de Hadamard en función del número de qubits del sistema y la cantidad de hilos paralelos.

en el calculo del error, se tomaron rotaciones en fases aleatorias $\phi \in [0, 2\pi)$, por lo que se realizan varias operaciones de punto flotante antes de llegar al resultado. De todas formas, se obtiene un error del orden de 10^{-20} para aquellos n en que la paralelización es aplicable.

CNOT

A continuación se caracterizara la ejecución de la compuerta CNOT, una de las fundamentales. Aquí, también se comparará el resultado de ejecutar la compuerta con el vector de estado esperado, es decir $\text{CNOT}|c, a\rangle = |c, \neg a\rangle \iff c = 1$. Se puede observar en la figura 5.6 como la paralelización es menos eficiente en la ejecución de la compuerta CNOT que en la inicialización. Hasta $n = 26$ se tiene que aún es más eficiente ejecutarla en serie que en 2 hilos, aunque por la tendencia de las curvas se ve que el tiempo de ejecución crece más rápido en el primer caso.

En la figura 5.7 se estudia el error de la compuerta, tanto numérico como algorítmico. Al ser una compuerta de dos qubits, puede verse que es más susceptible al problema de paralelizar para n pequeños, y recién para $n > 14$ se soluciona este problema

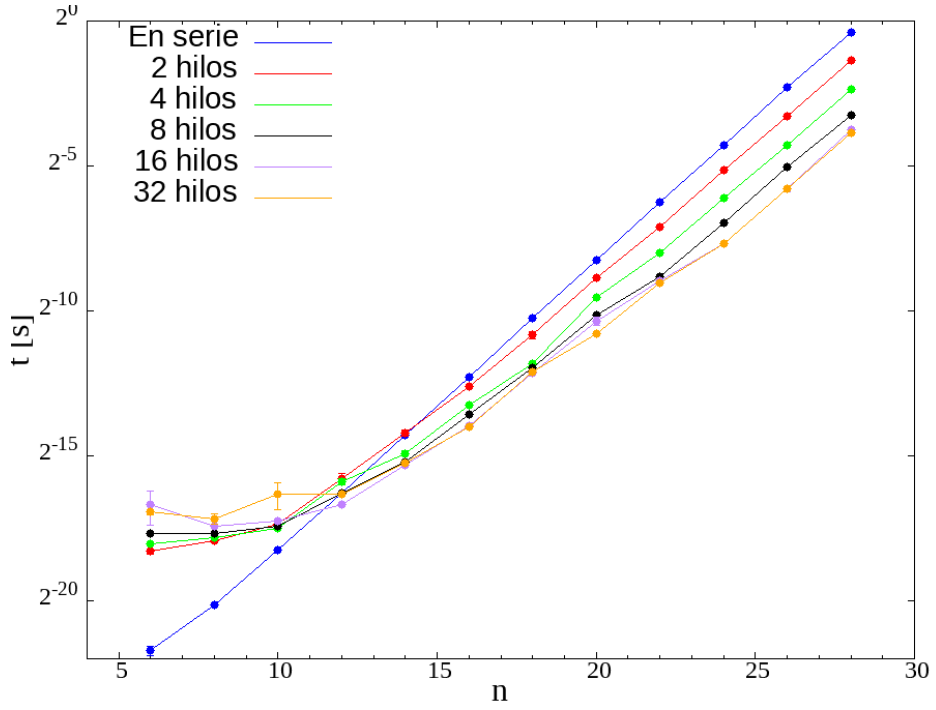


Figura 5.4: Tiempo de ejecución de la compuerta R(ϕ) en función del número de qubits del sistema y la cantidad de hilos paralelos.

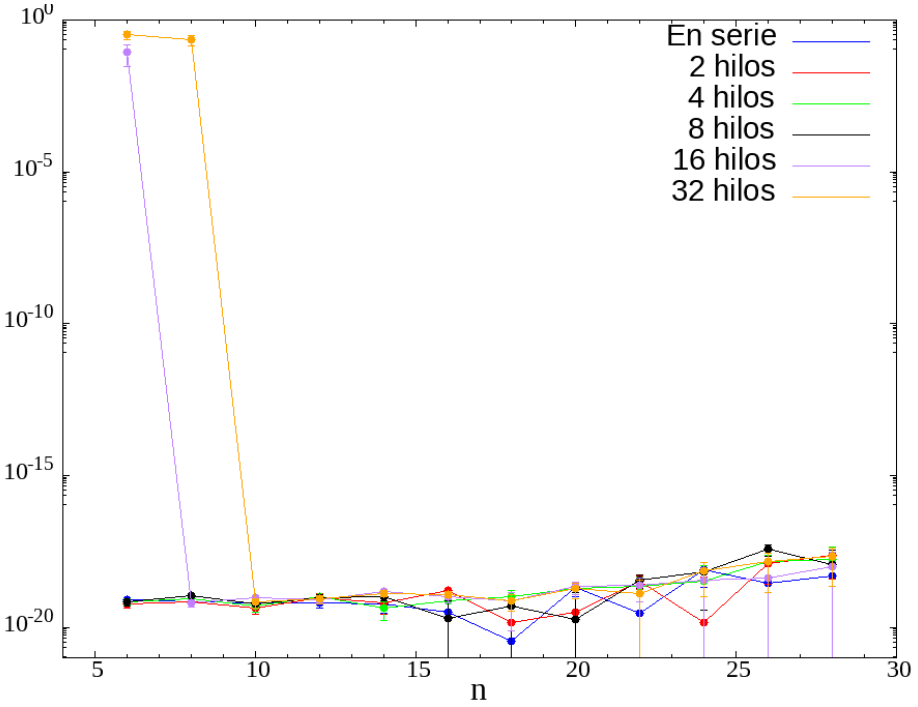


Figura 5.5: Error de R(ϕ) en función del número de qubits del sistema y la cantidad de hilos paralelos.

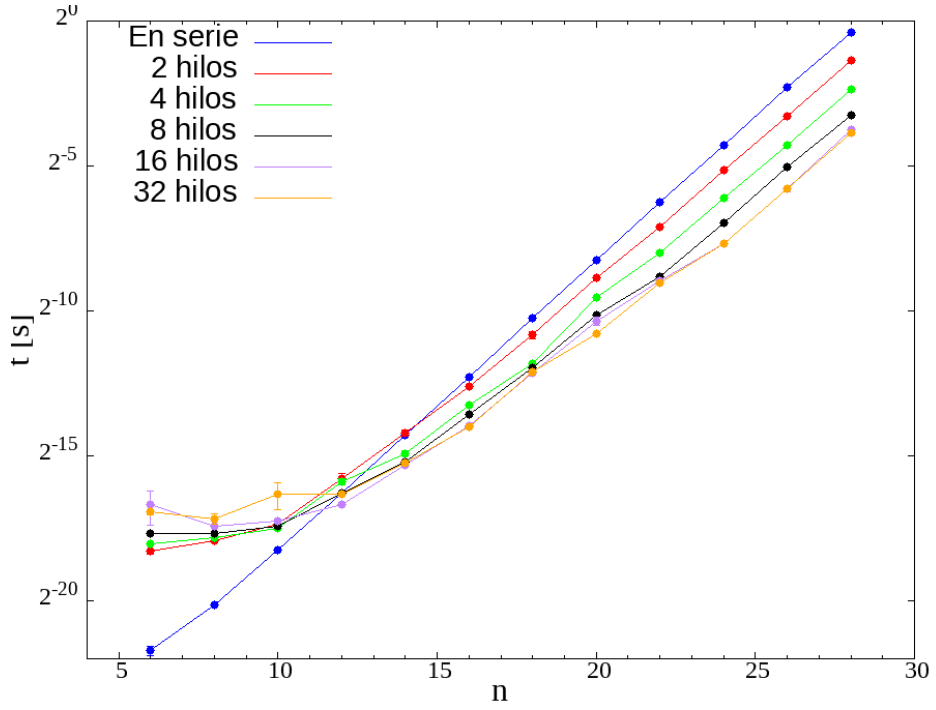


Figura 5.6: Tiempo de ejecución de la compuerta CNOT en función del número de qubits del sistema y la cantidad de hilos paralelos.

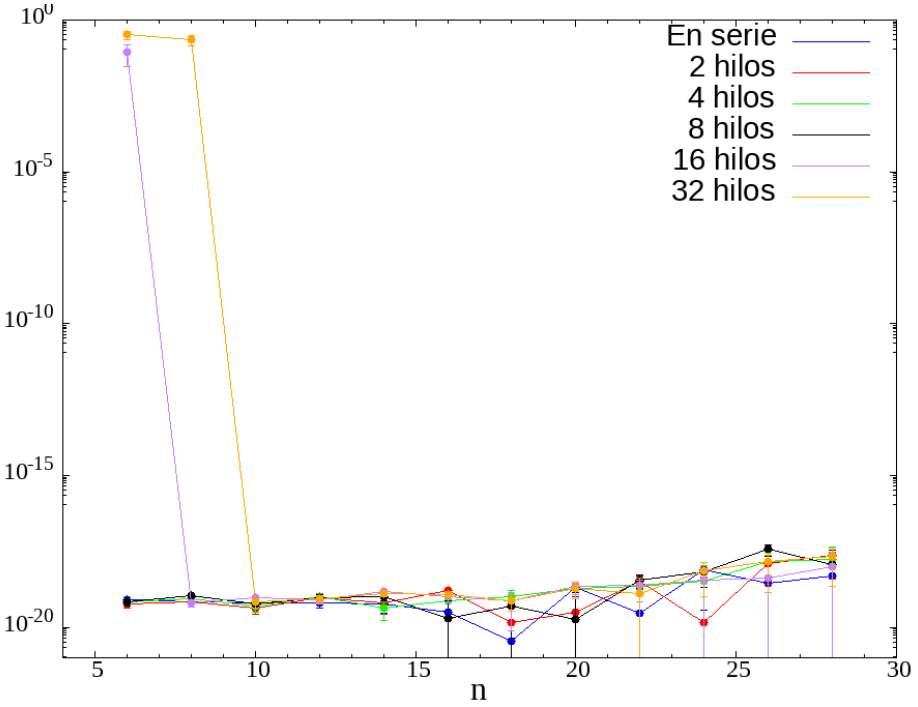


Figura 5.7: Error de CNOT en función del número de qubits del sistema y la cantidad de hilos paralelos.

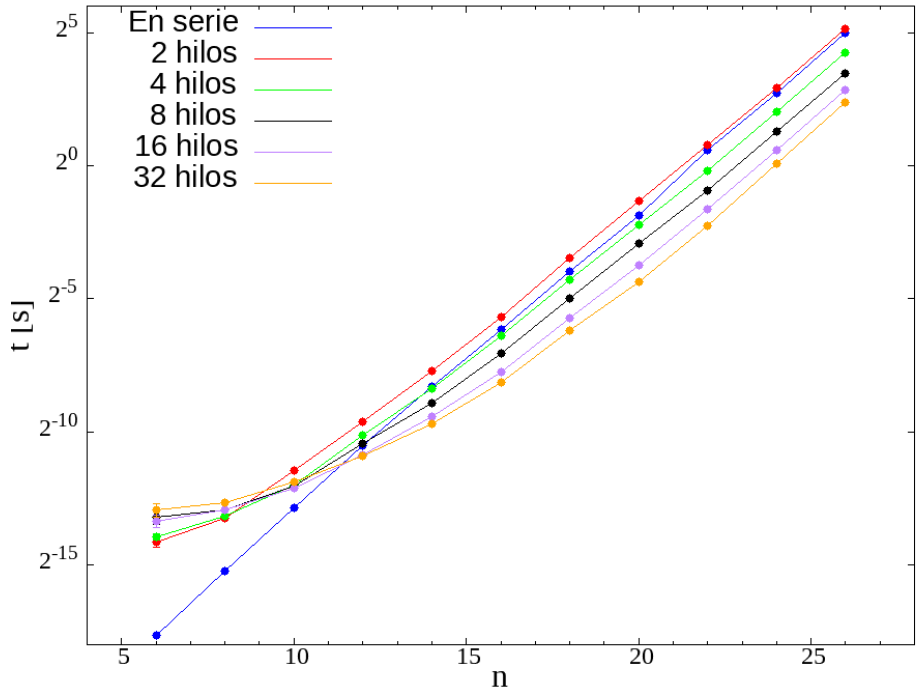


Figura 5.8: Tiempo de ejecución de la QFT en función del número de qubits del sistema y la cantidad de hilos paralelos.

5.3.3. Algoritmos Básicos

A continuación se estudiaran las operaciones básicas que se implementan mediante un circuito compuesto de las compuertas fundamentales de arriba. Esto abre el análisis de la complejidad algorítmica en término de número de compuertas, cuestión que es interesante ya que no es algo propio de implementaciones numéricas y es útil también en computadoras cuánticas.

QFT

La QFT presenta muchos matices para analizar. No sólo su desempeño en tiempo según la paralelización sino también, el número de compuertas fundamentales que se aplican durante su ejecución. Además, como vimos en la sección 4.4.6, su circuito puede simplificarse utilizando una versión aproximada de sí misma.

Primero, analizamos el tiempo de ejecución como en las compuertas anteriores en la figura 5.8, donde se observa con mucha claridad el comportamiento típico de la paralelización. La implementación en serie es claramente superior para tamaños pequeños, pero a medida que crece n se van produciendo cruces sucesivos entre la curva correspondiente a su tiempo de ejecución y las de las implementaciones paralelas.

En la figura 5.9 se muestra el error de aplicar sucesivamente la compuerta QFT y su inversa,

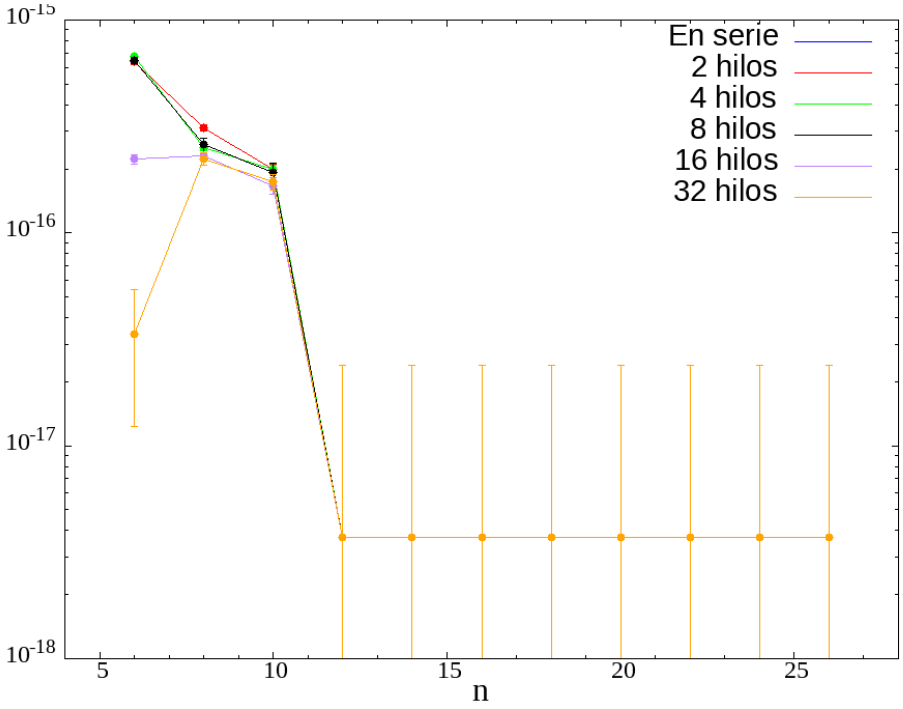


Figura 5.9: Error de QFT y su inversa en función del número de qubits del sistema y la cantidad de hilos paralelos.

lo cuál nos debe devolver al estado inicial.

A continuación analizaremos el comportamiento de la QFT aproximada a medida que se varía el parámetro k_{max}. Tomando $k_{max} = O(\log(n/\epsilon))$, se eligen dos valores distintos para ϵ. Esto produce valores de k_{max} en función de n que se muestran en la tabla 5.1. Aquí $k1$ y $k2$ se refieren a usar los k_{max} correspondientes a $\epsilon = 10^{-1}$ y $\epsilon = 10^{-3}$ respectivamente. Estos mismos valores se utilizaran cada vez que nos refiramos a k_{max}, en esta subsección y en la siguiente.

En la figuras 5.10 se muestra el número de compuertas para cada aproximación de QFT y de la exacta, mientras que en 5.11 y 5.11 se muestran el tiempo de ejecución de cada uno y el error de las implementaciones aproximadas con respecto a la exacta.

Se puede observar en la figura 5.10 como en las implementaciones aproximadas el número de compuertas comienza a alejarse de $O(n^2)$ como en el caso de la QFT exacta a medida que n crece. Del mismo modo, en 5.11 el tiempo de ejecución también empieza a mostrar diferencias.

Por último, se mide el error de estas implementaciones aproximadas con respecto a la QFT exacta. Se puede ver en 5.12 como este error se mantiene constante en n para cada k_{max}. El salto en la curva $e5$ se debe a que para $n < 12$ esta aproximación no desprecia ninguna compuerta, por lo que es exacta.

Si bien se observa una disminución significativa en el número de compuertas a medida que n

Exacto	$k1$	$k2$
6	5	6
8	6	8
10	6	10
12	6	12
14	7	13
16	7	13
18	7	14
20	7	14
22	7	14
24	7	14
26	8	14

Cuadro 5.1: Valores utilizados para k_{max} según el ϵ elegido, en función de n. $k1$ corresponde a $\epsilon = 10^{-1}$ y $k2$ a $\epsilon = 10^{-3}$.

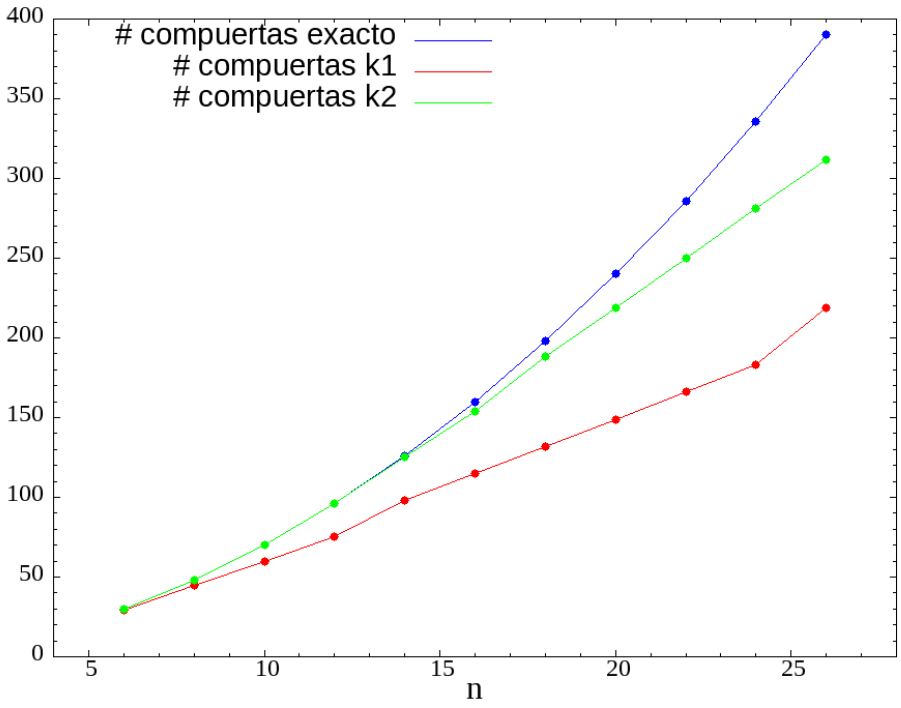


Figura 5.10: Número de compuertas fundamentales según k_{max} en función de n.

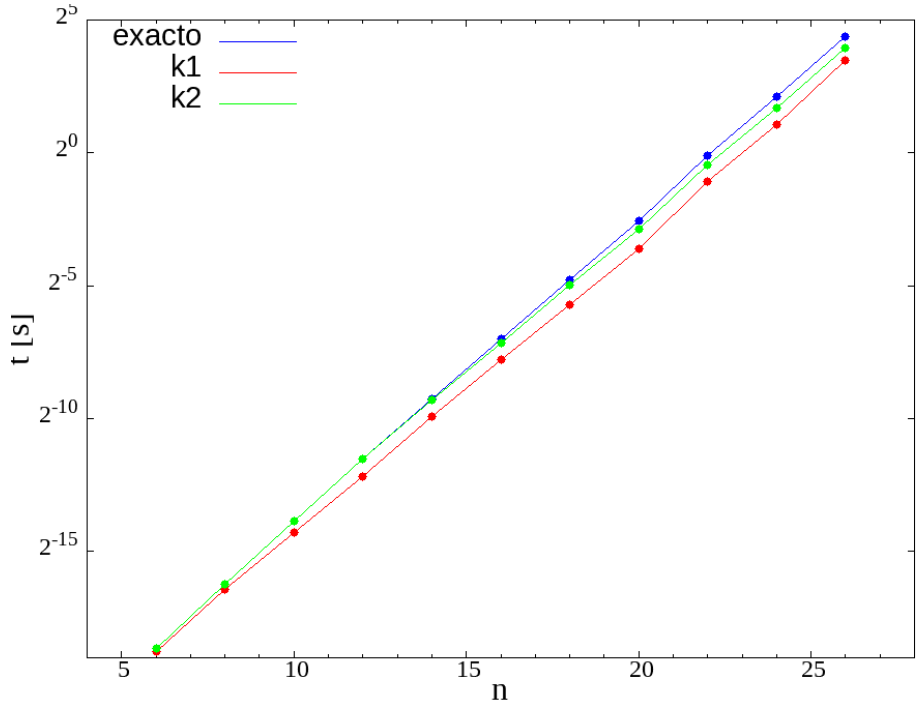


Figura 5.11: Tiempo de ejecución de la QFT exacta y aproximada en función del número de qubits del sistema (En serie).

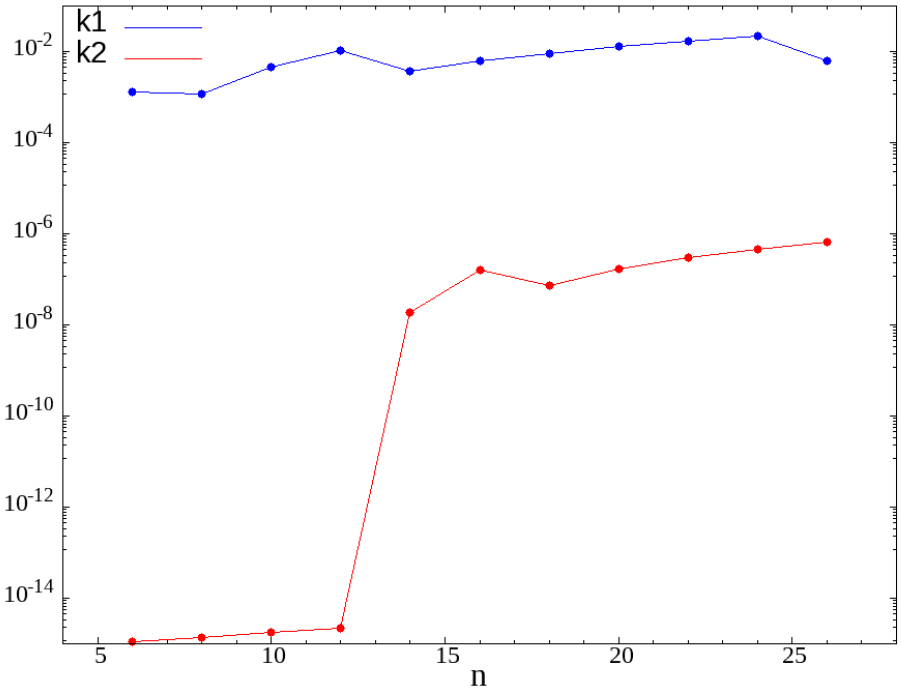


Figura 5.12: Error de las QFT aproximadas con respecto a la exacta en función del número de qubits del sistema.

crece hasta 26, es importante notar que en el algoritmo de estimación de fase la QFT sólo se aplica sobre registros parciales, no sobre todos los qubits. Esto significa que, aunque todos los qubits del sistema estén entrelazados y las distintas compuertas fundamentales del circuito de la QFT se aplican sobre el vector de estado de todo el sistema, la QFT como un todo es sólo sobre dicho registro parcial, por lo que no se aplican ni tantas compuertas ni se usan rotaciones tan pequeñas.

Por ejemplo, en la implementación de $2n + 2$ qubits de Takahashi, las QFT se aplican sobre el primer registro de n qubits, por lo que aunque el número total de qubits sea tan alto como 26, las QFT son sobre sólo n =12 qubits, y como se observa en la figura 5.10, la aproximación con $\epsilon = 10^{-3}$ todavía no ha despreciado ninguna compuerta.

ϕ Adder

En esta subsección se estudia la operación de adición en el espacio de Fourier, que se utiliza en ambas implementación presentadas en el capítulo anterior. Esta compuerta es similar a la QFT al basarse también en rotaciones, por lo que se procederá a su estudio de manera similar. Aquí, el error se computa de la siguiente forma: se inicializa el estado del sistema a un estado no entrelazado $|b\rangle$ con b tomado al azar entre $0, \ldots, 2^n - 1$ y se le aplica la compuerta $\phi ADD(a)$ para cierto a también al azar entre $0, \ldots, 2^n - 1$. Entonces, se genera otro vector con el resultado esperado, el estado $|(a + b) \bmod 2^n\rangle$, y se computa la proyección.

Primero, en las figuras 5.13 y 5.14 se muestran el tiempo de ejecución y el error respectivamente , cuyos comportamientos son los esperados y muy similares a los de la QFT.

Como la compuerta $\phi ADD(a)$ también se basa en rotaciones, puede aproximarse como la QFT despreciando las rotaciones R_k con $k > k_{max}$. En las figuras 5.15, 5.16 y 5.17 se estudia el desempeño de estas aproximaciones en número de compuertas, tiempo de ejecución y error relativo, respectivamente

Al igual que en la QFT, se observa un cambio en el comportamiento asintótico pero de igual forma, la mejora en la complejidad de la compuerta no compensa el aumento del error para los n con los que se trabajará a partir de ahora, por lo que se seguirá tomando $k_{max} = n$ en las compuertas de adición modular.

Comparador COMP

La compuerta COMP que se utiliza en la implementación de Takahashi para decidir si aplicar $\phi ADD(N - a)$ o $\phi SUB(a)$ se basa en compuertas de negación controladas, por lo que no puede aproximarse con rotación. De esta manera, se mostrará el comportamiento de la misma en cuanto a tiempo, error y cantidad de compuertas en las figuras 5.18, 5.19 y 5.20 respectivamente. Como

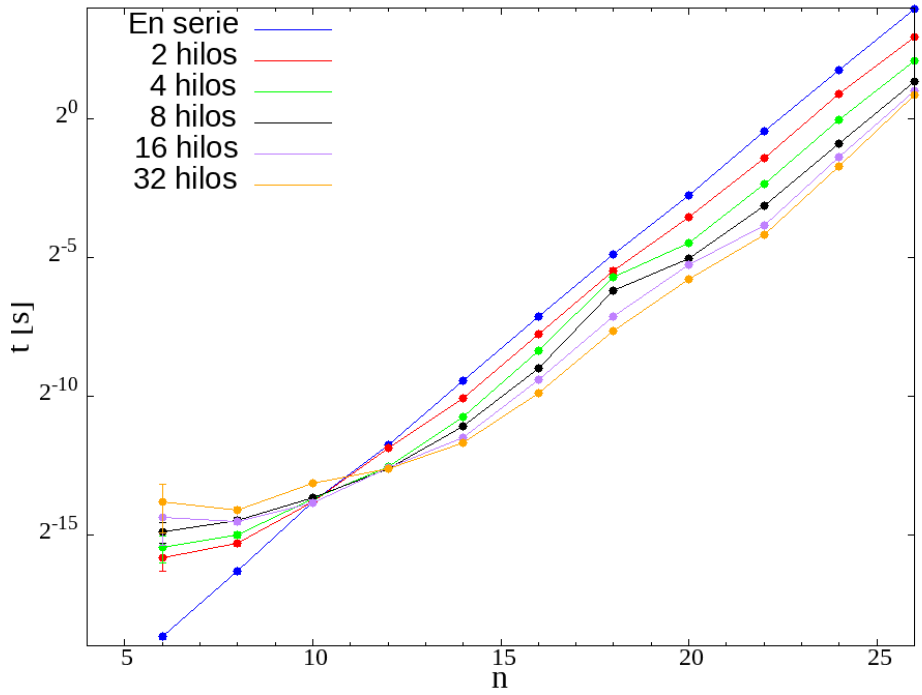


Figura 5.13: Tiempo de ejecución de la compuerta ϕADD en función del número de qubits del sistema y la cantidad de hilos paralelos.

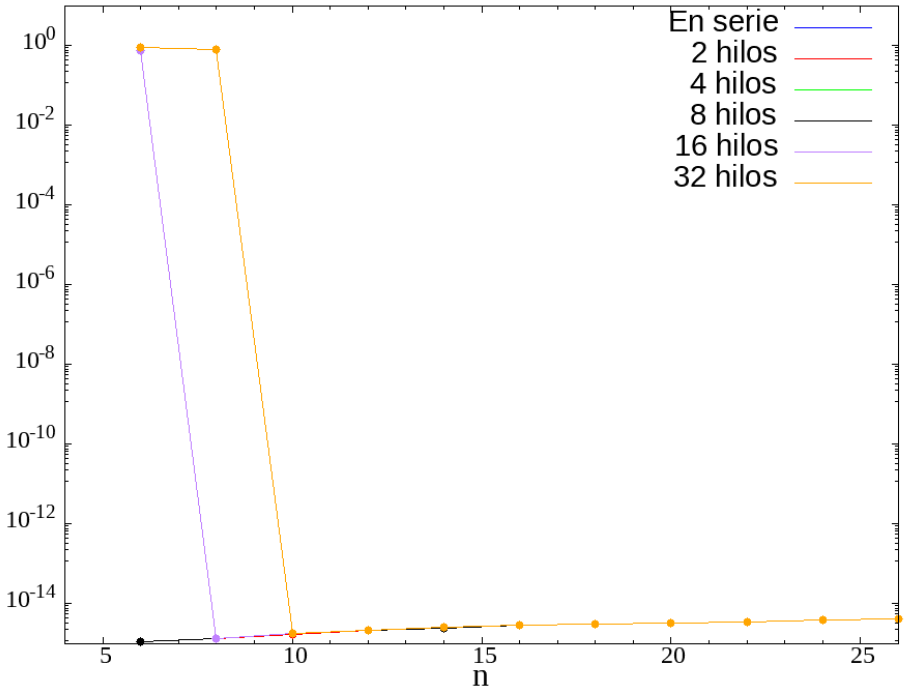


Figura 5.14: Error de aplicar $\phi ADD(a)$ a un estado $|b\rangle$ en función del número de qubits del sistema y la cantidad de hilos paralelos.

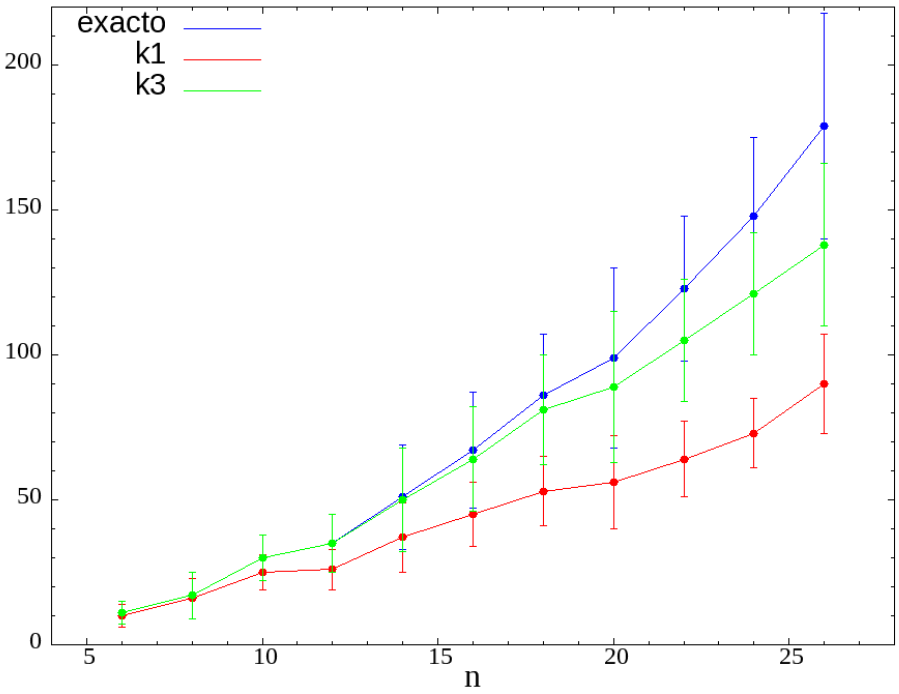


Figura 5.15: Número de compuertas de ϕADD según k_{max} en función del número de qubits del sistema n.

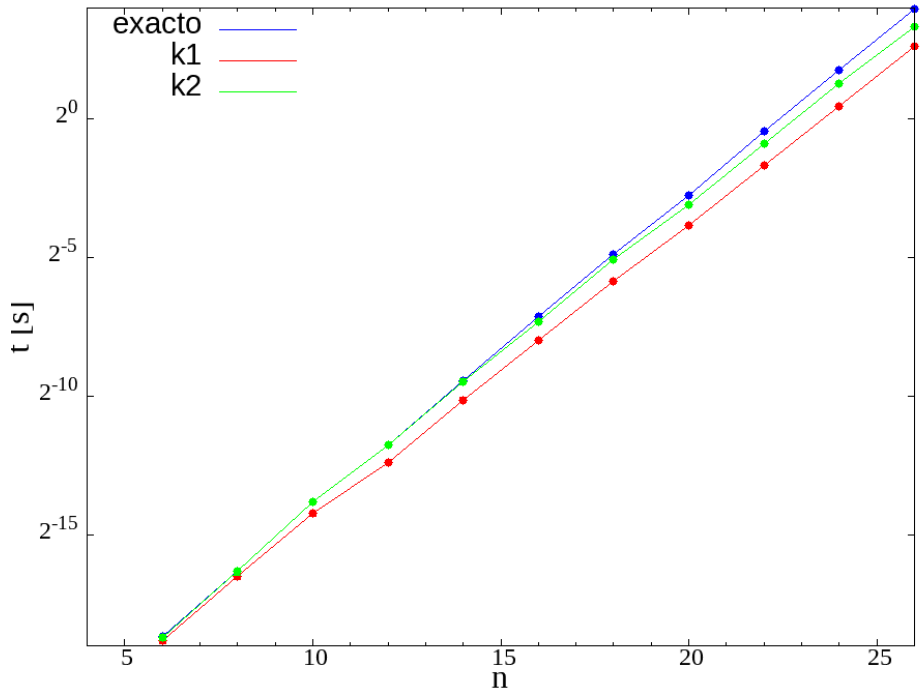


Figura 5.16: Tiempo de ejecución de la compuerta ϕADD según k_{max} en función del número de qubits del sistema

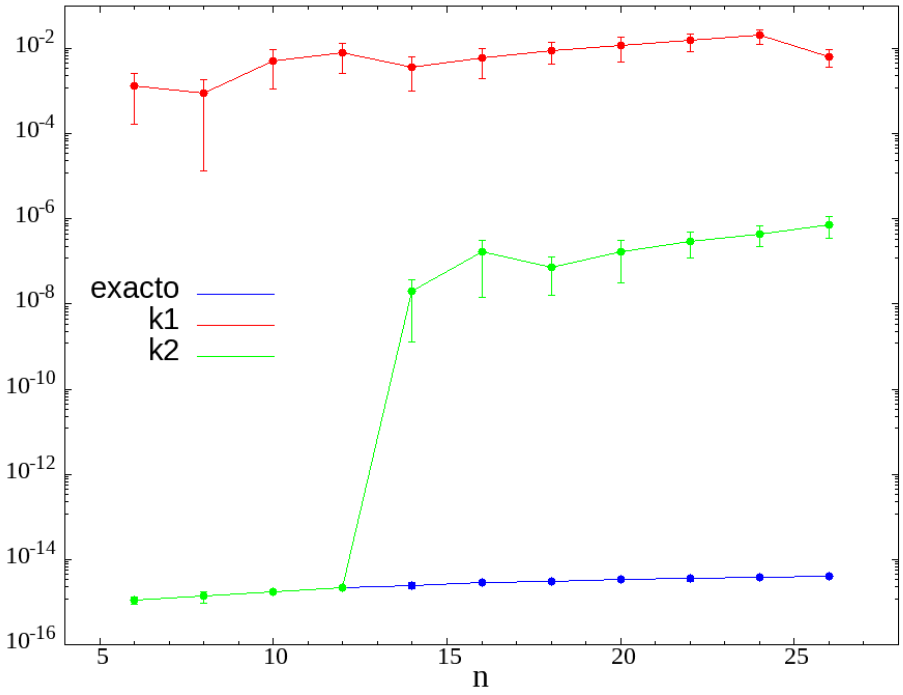


Figura 5.17: Error de las aproximaciones k_{max} con respecto a la compuerta ϕADD exacta.

esta compuerta requiere el total de los $2n+2$ qubits necesarios para el algoritmo de Shor, se estudia un rango de n más chico.

Para el calculo del error se inicializa el estado del primer y el segundo registro a estados no entrelazados $|x\rangle$ y $|b\rangle$ con x, b tomado al azar entre $0, \ldots, 2^n - 1$. Luego, se le aplica la compuerta COMP(a) para cierto a también al azar entre $0, \ldots, 2^n - 1$. Luego, se compara el estado del sistema resultante con el esperado considerando si se debe invertir el último qubit, lo que ocurre cuando $b < a$ y el qubit de control del primer registro es 1.

Los gráficos del error y tiempo de ejecución de COMP son similares a los de las demás compuertas, aunque con mayores barras de error. Esto último se debe a que el número de compuertas varía según el a con que se quiere hacer la comparación, como se observa en las barras de error de la figura 5.20. También se observa el comportamiento lineal en n de la compuerta COMP.

Adición Modular

Las compuertas de adición modular son, debido a su complejidad, la parte central de ambos algoritmos. A continuación se analizará comparativamente el desempeño de ambas, teniendo en cuenta la ventaja de la adición modular de Takahashi de necesitar un qubit ancilla menos, lo cuál se reflejará sobre todo en su tiempo de ejecución. En la figura 5.21 se muestra el tiempo de ejecución en función de n para ambos algoritmos, tanto para la implementación en serie como para

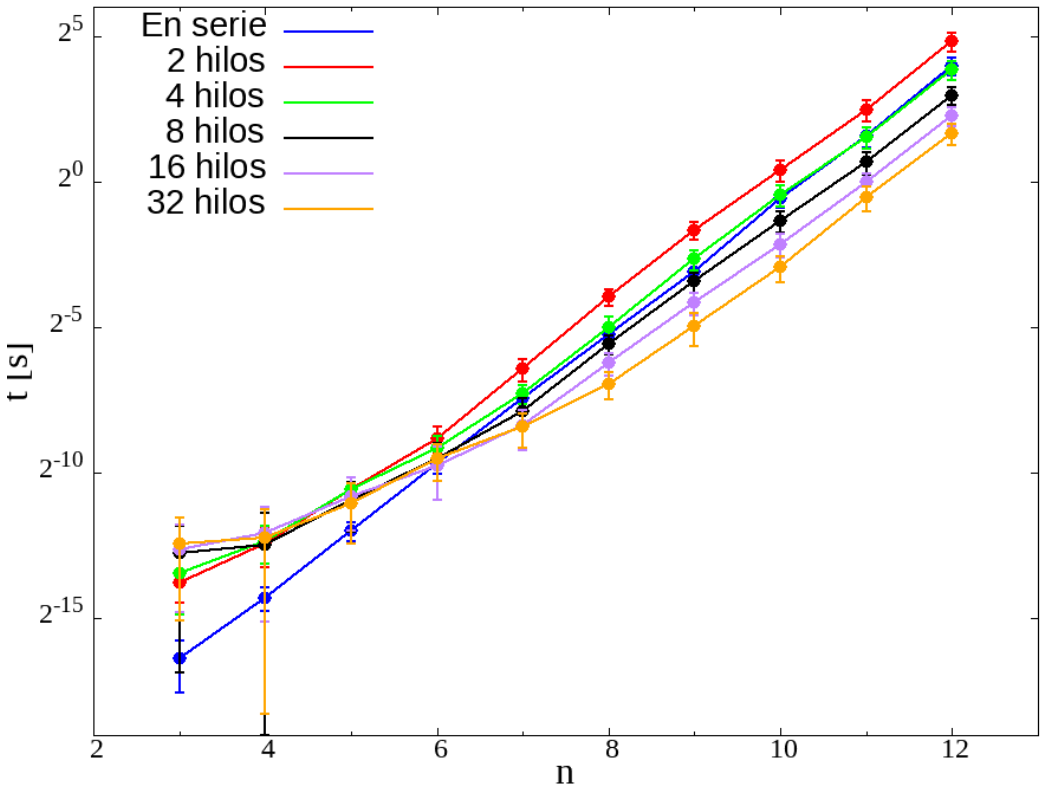


Figura 5.18: Tiempo de ejecución de la compuerta COMP en función del número de qubits del sistema y la cantidad de hilos paralelos.

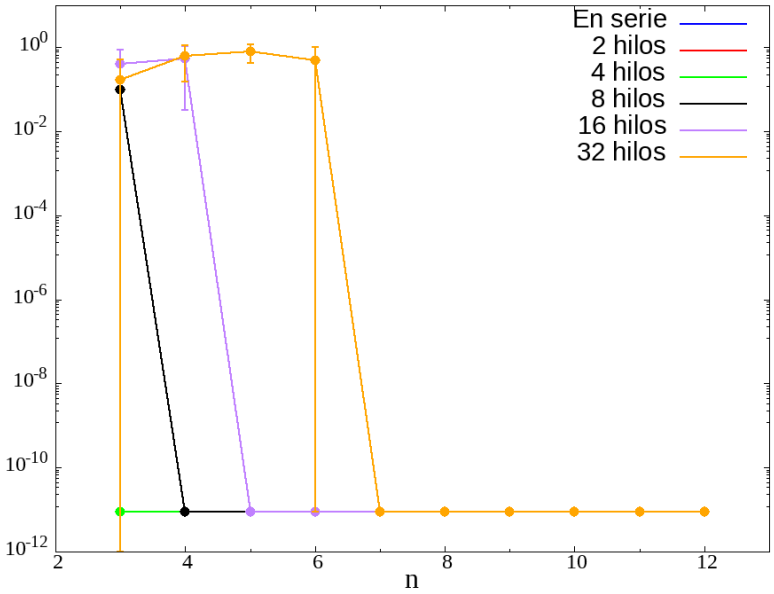


Figura 5.19: Error de aplicar COMP(a) a un estado $|b\rangle$ en función del número de qubits del sistema y la cantidad de hilos paralelos.

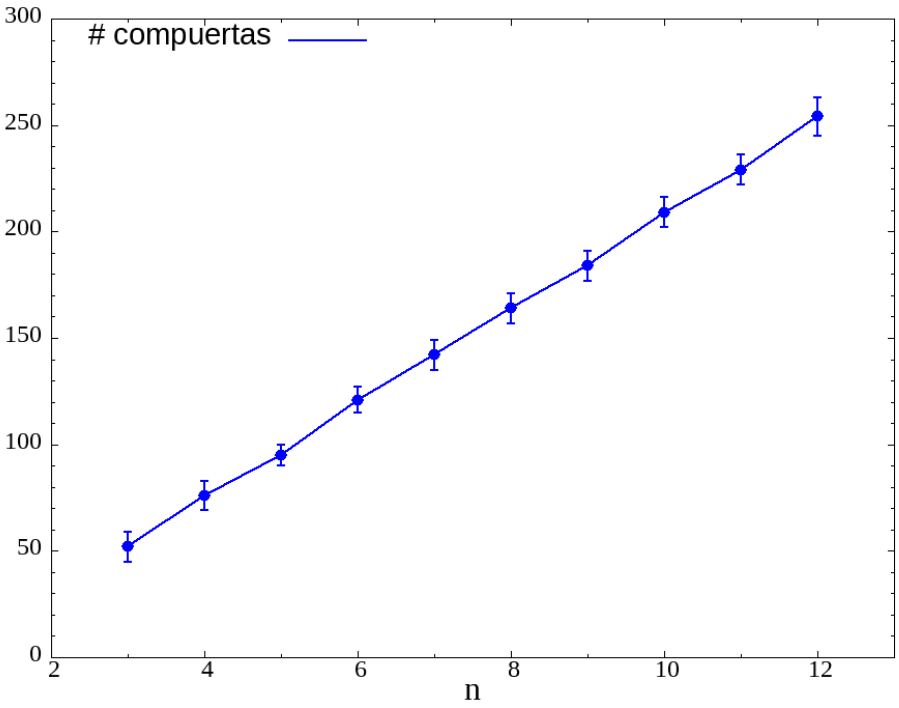


Figura 5.20: Número de compuertas de COMP y su desviación estándar en función de n.

la implementación en 32 hilos paralelos.

Puede observarse que la adición modular presenta el comportamiento característico según sea en serie o en paralelo, como así también que para n grandes la implementación de Beauregard tarda el doble debido al qubit extra. Notemos tambien que en las implementaciones en paralelo para $n < 7$, la implementación de Beauregard es más rápida a pesar de dicho qubit extra.

Lo más interesante de estudiar es como se comparan ambas implementaciones entre sí según su número de compuertas fundamentales. En la figura 5.22 se muestran ambos datos en función de n.

Este gráfico es de los más ilustrativos del trabajo, ya que permite observar y ejemplificar varios comportamientos interesantes en relación a la complejidad algorítmica de ambas implementaciones. Junto con las curvas de la media del número de compuertas con la desviación estándar como barras de error, se grafican los comportamientos asintóticos para n grandes y chicos. Como se puede observar en las figuras 5.10, 5.22 y 5.20 para n chicos domina el comportamiento lineal en n y esto se traslada a la adición modular.

Además, notemos como se produce un cruce entre el número de compuertas fundamentales para cada implementación. Esto se debe a que el algoritmo de Beauregard utiliza más las compuertas QFT y $\phi ADD(a)$ cuyo comportamiento es cuadrático, mientras que Takahashi utiliza la compuerta COMP que tiene comportamiento lineal. Esta última, al necesitar un mayor número de compuertas fundamentales para n chico, es lo que impulsa el total de compuertas por encima

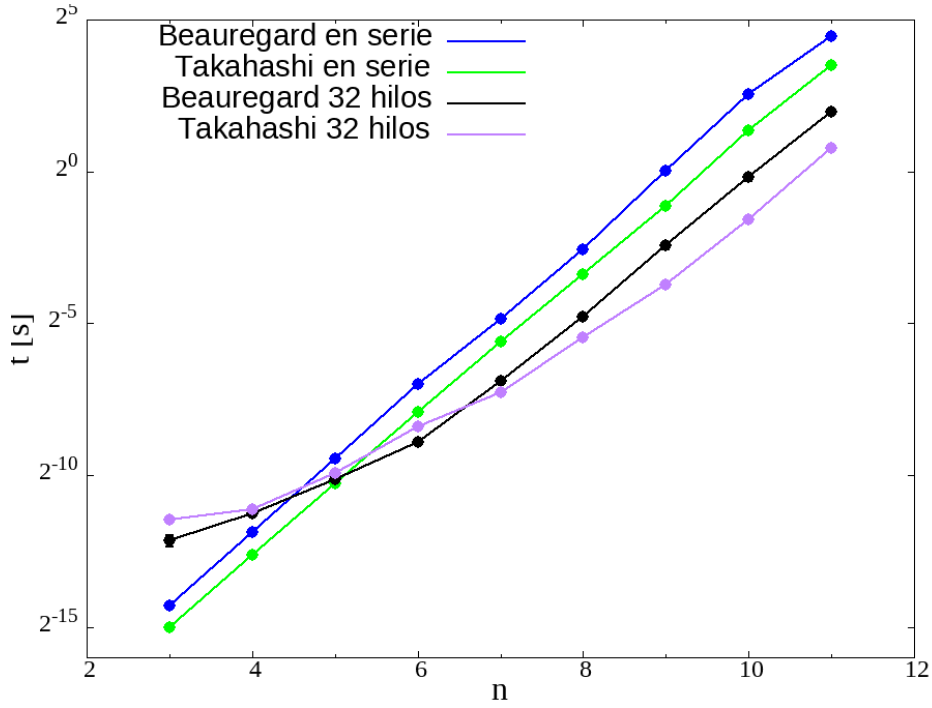


Figura 5.21: Tiempo de ejecución de las compuertas de adición modular en función del número de qubits y la cantidad de hilos paralelos.

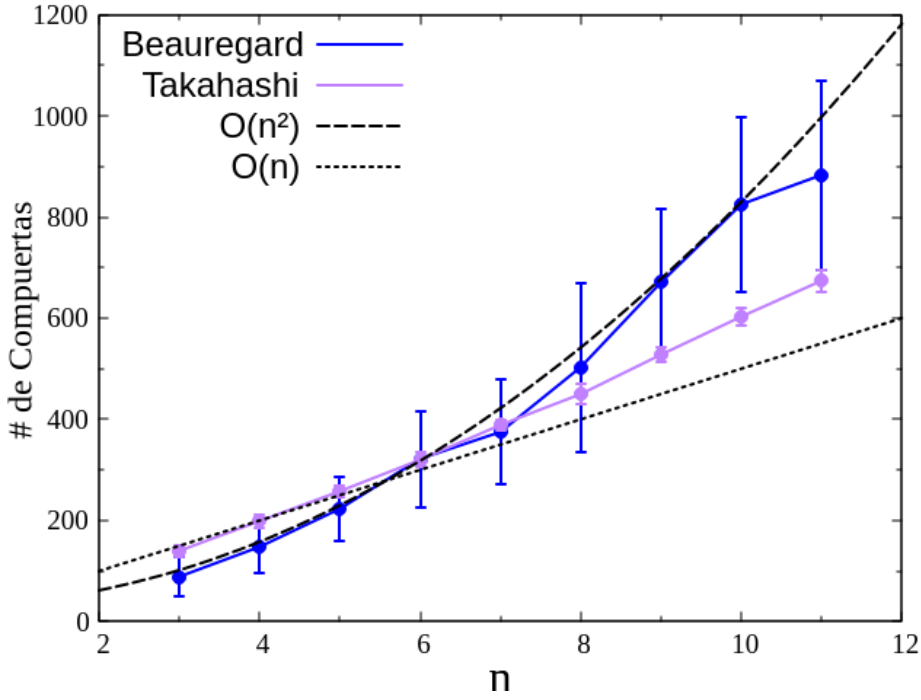


Figura 5.22: Número de compuertas fundamentales necesarias para la adición modular en función de n, incluyendo curvas de tendencia lineal y cuadrática.

de la implementación de Beauregard, aunque a medida que crece n esto se invierte debido al ya mencionado mayor uso de esta última implementación de compuertas con complejidad cuadrática.

Por último, observemos como la desviación estándar de Beauregard es mucho mayor que la de Takahashi. Esto se debe al mayor uso de compuertas $\phi ADD(a)$, cuyo uso de compuertas fundamentales depende fuertemente del valor de a. Esto repercute no sólo en mayores barras de error, si no también en una menor suavidad en la curva: los valores de la media presentan saltos bruscos. La solución a esta falta de suavidad es aumentar la estadística sobre la cuál se calcula cada media y cada desviación estándar. La implementación de Takahashi, aunque utiliza las compuertas COMP(a) y $\phi ADD(a)$ cuyo número de compuertas depende de a, utiliza sólo tres de ellas, a diferencia de las cinco que utiliza Beauregard.

5.3.4. Algoritmo de Shor

Una vez ya hemos estudiado y caracterizado las compuertas en las que se basa la subrutina para hallar el orden del algoritmo de Shor, procederemos ahora a analizarla como un todo. En esta subsección se estudiará el desempeño en tiempo de las distintas implementaciones en serie y en paralelo, el número de compuertas, y también se caracterizará el error con respecto a variaciones en t, el cuál como vimos en la sección 4.2.3 parametriza la exactitud con la cuál se estima la fase $\varphi_j \sim j/r$. Luego, se mostrará el histograma de probabilidad para varios valores de N y a, y su dependencia en t.

En línea con los análisis que hemos hecho hasta ahora, comenzamos con la figura 5.23 en la que se muestra el tiempo de ejecución de todo el algoritmo de Shor, el cuál presenta como era esperado un comportamiento muy similar al de las compuertas de adición modular.

Complejidad Algorítmica: Número de Compuertas

El estudio del comportamiento del número de compuertas fundamentales en función del tamaño de la entrada n es el más importante, ya que es el que puede ser de utilidad en una posible aplicación de estas implementaciones del algoritmo de Shor en una computadora cuántica. En las figuras 5.24 se muestra el número de compuertas en función de n.

Observemos como se presenta un comportamiento cercano al esperado por su complejidad $O(n^3 \log n)$, incluso sistema de tamaño moderado. También allí se muestra una ampliación a los n más chicos para poder apreciar mejor como en este caso el algoritmo de Takahashi, en concordancia con la figura 5.22, precisa de más compuertas. Luego, debido a que como ya dijimos el algoritmo de Takahashi utiliza menos compuertas con complejidad cuadrática, el número de compuertas crece más lento que en la otra implementación, aunque gracias a la línea de tendencia podemos ver que ambas siguen un comportamiento similar al de su complejidad.

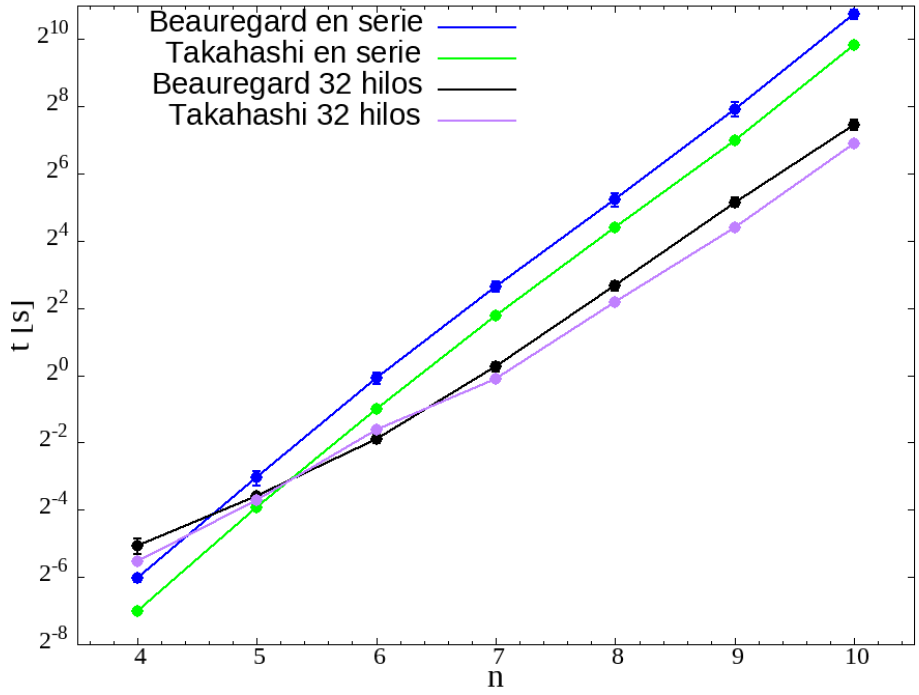


Figura 5.23: Tiempo de ejecución de la subrutina para hallar el orden del algoritmo de Shor en función del número de qubits y la cantidad de hilos paralelos.

Ahora, analizaremos la dependencia del número de compuerta a medida que se varía el parámetro t, para varios N y a seleccionados. En las figuras 5.25 se puede observar como el aumento del número de compuertas es linea en t. Además, en los valores para los que una implementación precisa más compuertas que la otra, esta diferencia sólo aumenta ligeramente con t.

Probabilidades de Medición

Como se deduce en la ecuación (4.62), la probabilidad de medir un valor $c = 0, \ldots, 2^t$ esta dada por

$$P(c) = \frac{1}{qM} \left| \sum_{d=0}^{M-1} \zeta^d \right|^2 \tag{5.27}$$

donde $q = 2^t$, $M \sim q/r$ y $\zeta = e^{(2\pi i\ cr/q)}$ y se tiene, como $\zeta \sim 1 \iff c/q \sim j/r$, que

$$P(c) \sim \frac{1}{r}\delta(c - [qj/r]) \tag{5.28}$$

En las figuras 5.26 se muestran la distribuciones de probabilidad para los valores posibles de c, para distintos N, a y t. Puede verse en las distribuciones de probabilidad los claros picos correspondientes a los $c \sim qj/r$.

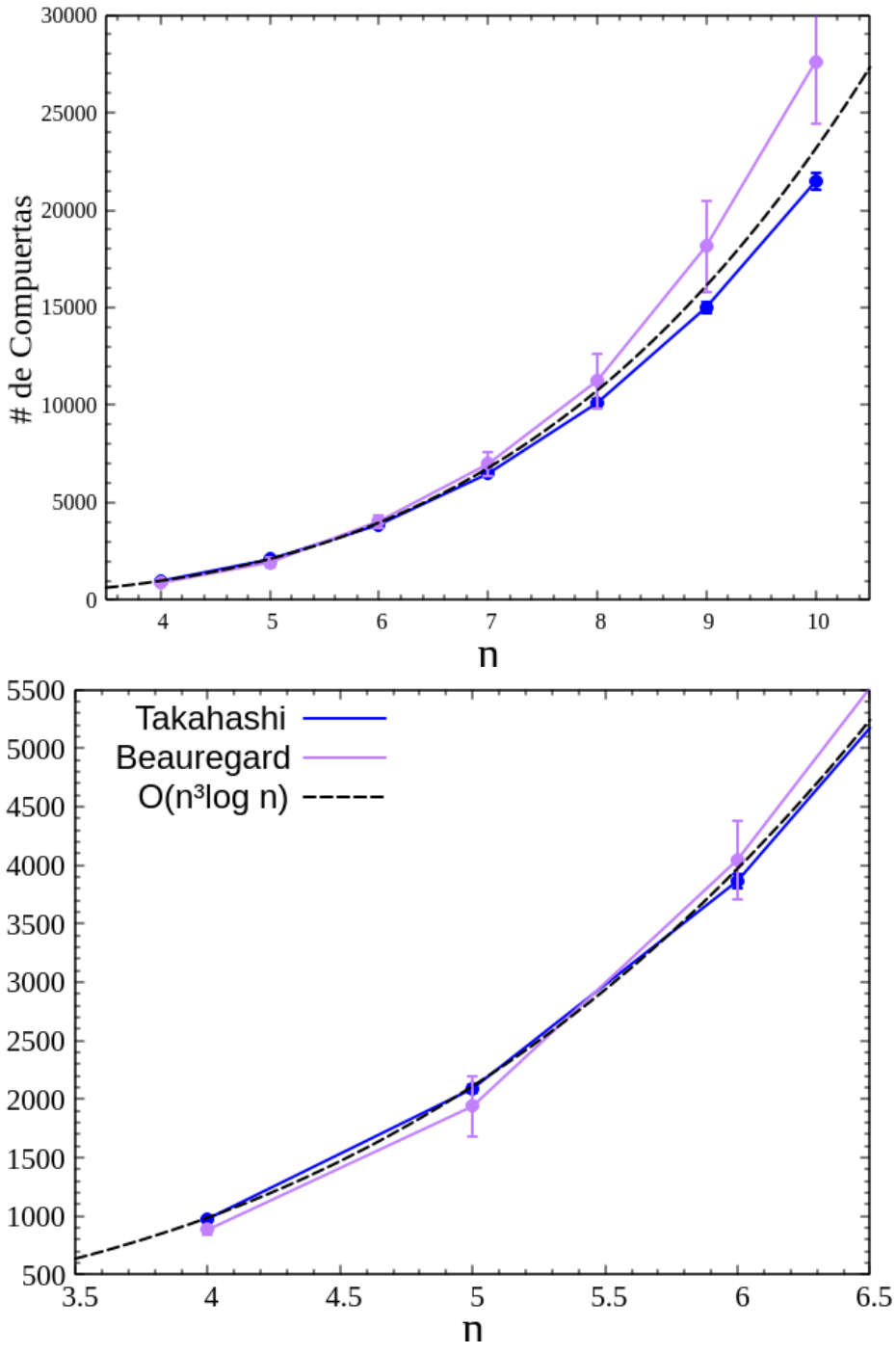
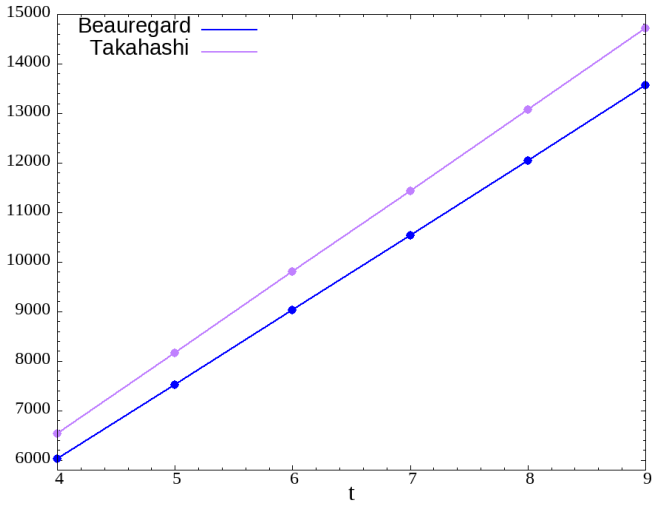
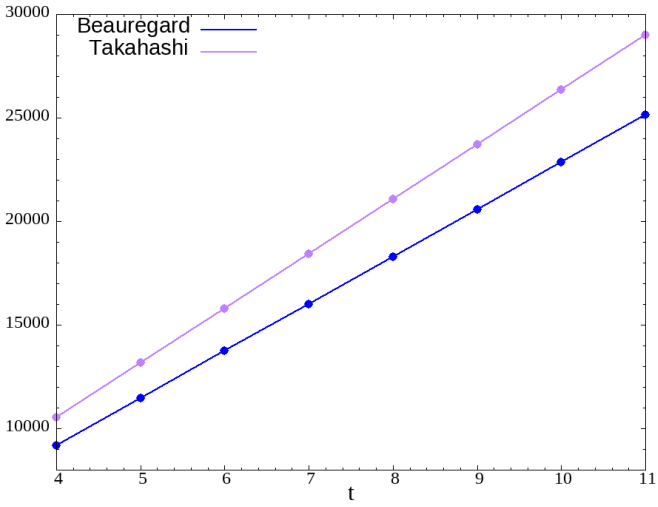


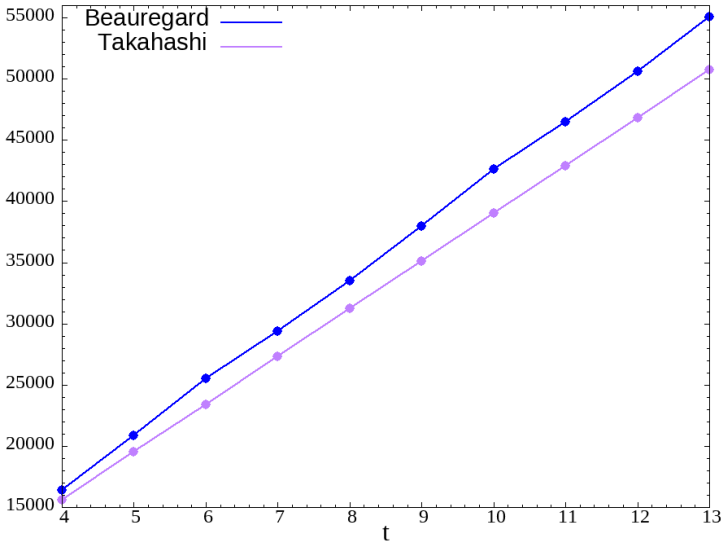
Figura 5.24: Número de compuertas fundamentales del algoritmo de Shor en función de n, mostrando la tendencia $O(n^3 \bmod n)$ esperada.



(a) $N = 15 \quad a = 2$.

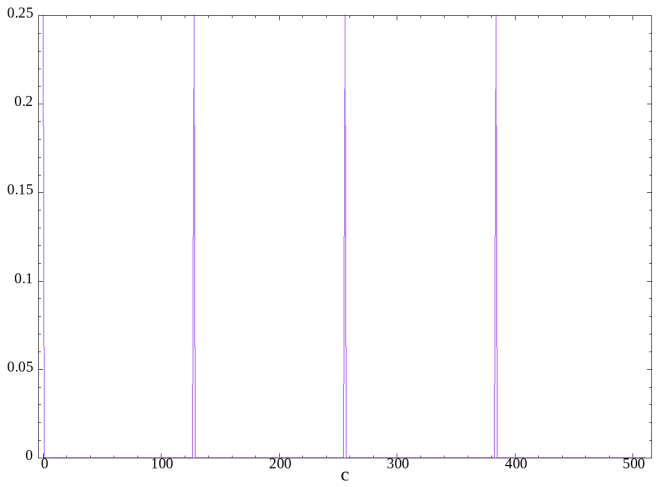


(b) $N = 21 \quad a = 11$.

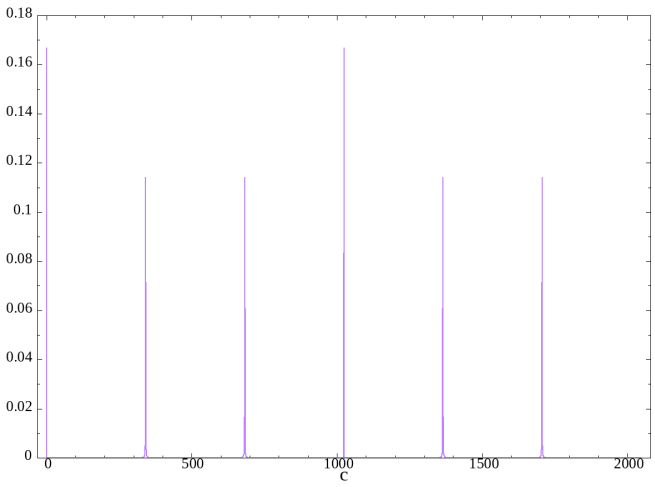


(c) $N = 55 \quad a = 13$.

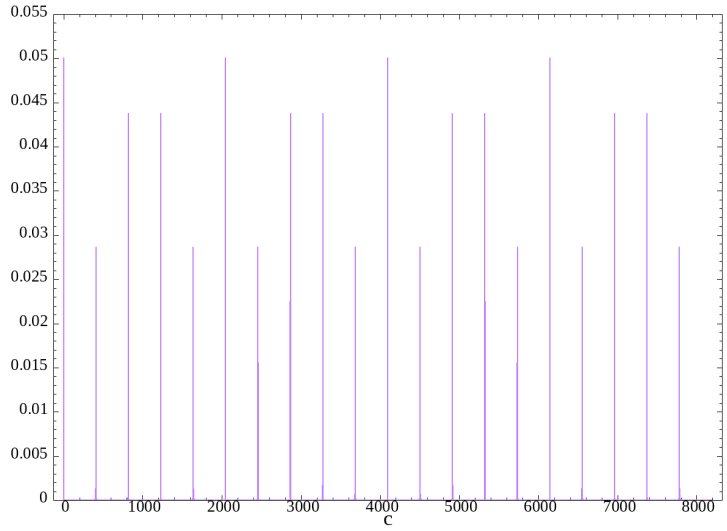
Figura 5.25: Número de compuertas fundamentales en función del parámetro de exactitud en la medición de la fase t.



(a) $N = 15 \quad a = 2 \quad t = 9$.



(b) $N = 21 \quad a = 11 \quad t = 11$.



(c) $N = 55 \quad a = 13 \quad t = 13$.

Figura 5.26: Probabilidad de medición de $c = 0, \ldots, 2^t$.

Divergencia de Kullback-Leibler

Como una medida del error de la distribución de probabilidad obtenida mediante el algoritmo de Shor, se la comparará con el valor teórico de la ecuación (5.27) utilizando la Divergencia de Kullback-Leibler [22], que se muestra en la ecuación (5.29). Esta es una medida no simétrica de la similitud o diferencia entre dos distribuciones de probabilidad P y P_0, que se corresponden con la distribución calculada y la teórica respectivamente. Cuanto más cercana a cero sea, más similares son P y P_0.

$$D_{KL}(P||P_0) = \sum_i P(i)\ \log\Big(\frac{P(i)}{P_0(i)}\Big) \tag{5.29}$$

En las figuras 5.27, se muestra $D_{KL}(P||P_0)$ para los casos anteriores, en función de t.

Excepto en el caso de $N = 15$, ambas implementaciones presentan divergencias similares, y estas se estabilizan al elegirse t con $q >> r$.

También, es interesante ver como varía cada término de la suma de la ecuación (5.29) que se corresponde a cada $c = i$. En las figuras 5.28 se muestra en escala logarítmica el término $P(c)\ \log\Big(\frac{P(c)}{P_0(c)}\Big)$ para todos los valores de c. Se observa como para $N = 15$ la distribución de picos es muy similar a los picos de probabilidad de la figura 5.26a. En los demás gráficos se presenta una distribución de picos pero también valles muy profundos, aunque de la misma forma los picos se producen cuando $c \sim qj/r$.

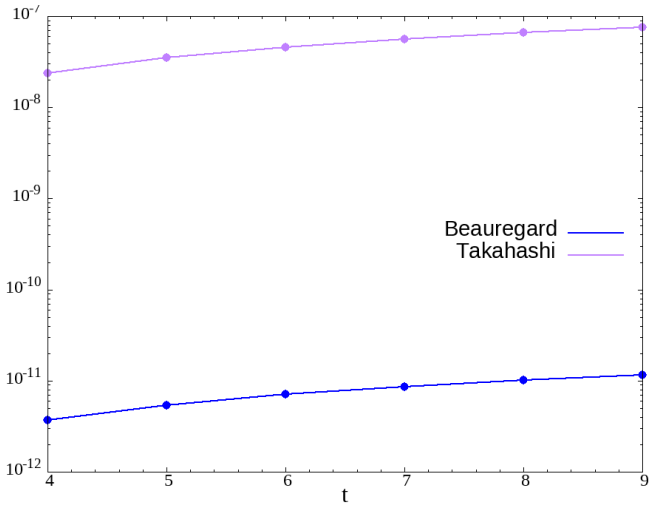
Uso de los resultados en el post procesamiento clásico

La subrutina cuántica del algoritmo de Shor precisa el posterior procesamiento de sus resultados para hallar el orden r de $a^x \bmod N$, y luego a partir de él realizar la factorización en números primos. Por ello, a pesar de como vimos en la subsección anterior la probabilidad de medir $c \sim qj/r$ son altas, debemos asegurarnos que en la práctica estos resultados sean útiles. Un ejemplo de medición teóricamente correcta pero inútil es medir $c = 0$, o sea, $j = 0$, pues puede corresponder a cualquier r.

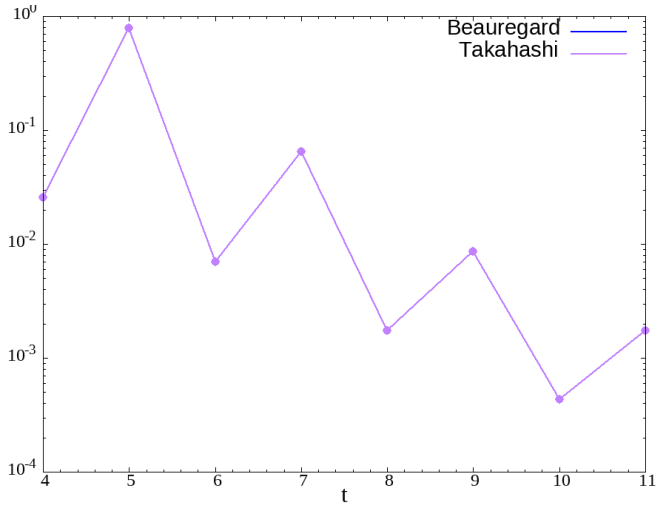
En las figuras 5.29 se muestra la probabilidad de medir algún c tal que $c = [qj/r] \quad \forall j \neq 0$, donde $[x]$ indica el entero más próximo. Esto podría considerarse como la probabilidad de éxito de la subrutina cuántica, aunque medir un c así no nos asegure poder obtener r.

Como vemos, la probabilidad de éxito es superior al 60 % para los casos expuestos aquí. Notemos como para t chico esta crece, pero esto es debido a que $q = 2^t$ se vuelve pequeño comparado con r por lo que $c = [qj/r]$ termina incluyendo a casi todos los c. Por ejemplo, $13^x \bmod 55$ tiene $r = 20$, y para $q = 2^4 = 16$ cada c cumple la condición anterior por lo su probabilidad se suma.

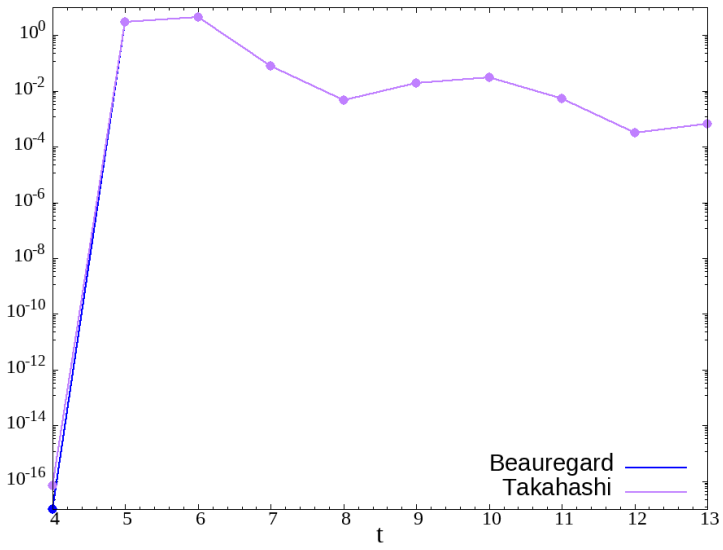
Por otro lado, aclaremos también que la probabilidad restante, es decir, la probabilidad de medir $c \neq [qj/r]$ no significa ruido, ya que la probabilidad crece con c cerca de qj/r. En particular, en el caso del ejemplo anterior, se tiene que la probabilidad de medir $c = [qj/r]$ con $j = 1$, o sea,



(a) $N = 15 \quad a = 2$.

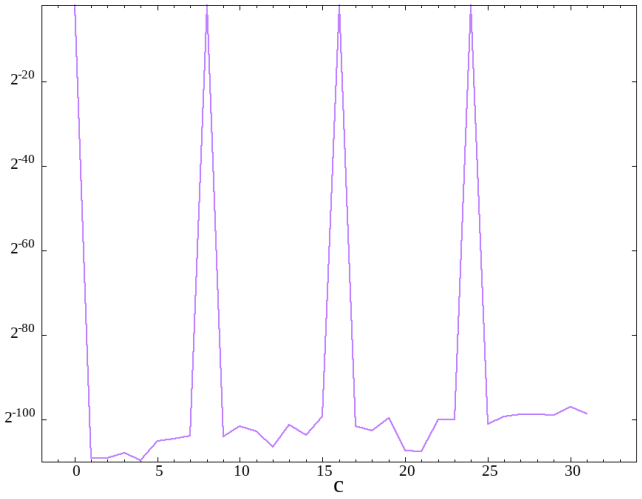


(b) $N = 21 \quad a = 11$.

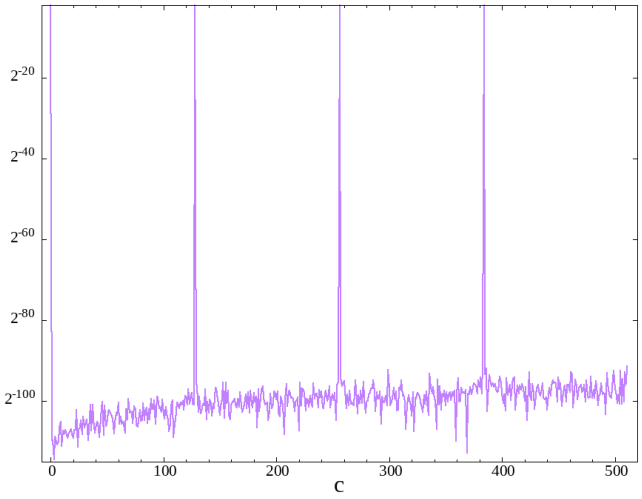


(c) $N = 55 \quad a = 13$.

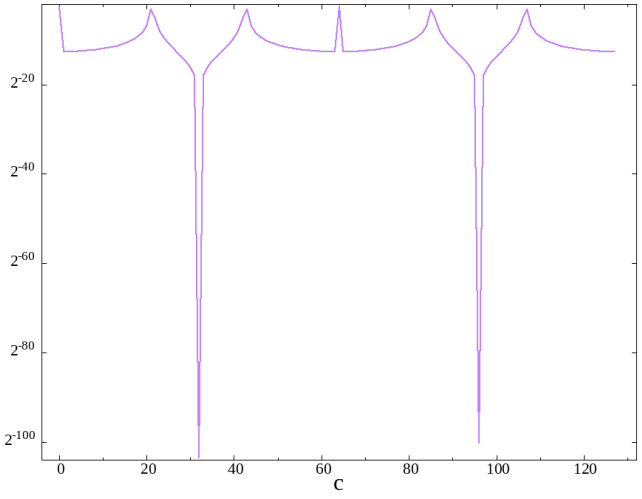
Figura 5.27: Divergencia de Kullback-Leibler de las distribuciones de probabilidad obtenidas en las simulaciones con respecto a la distribución de probabilidad teórica del algoritmo de Shor, en función del parámetro de exactitud en la medición de la fase t.



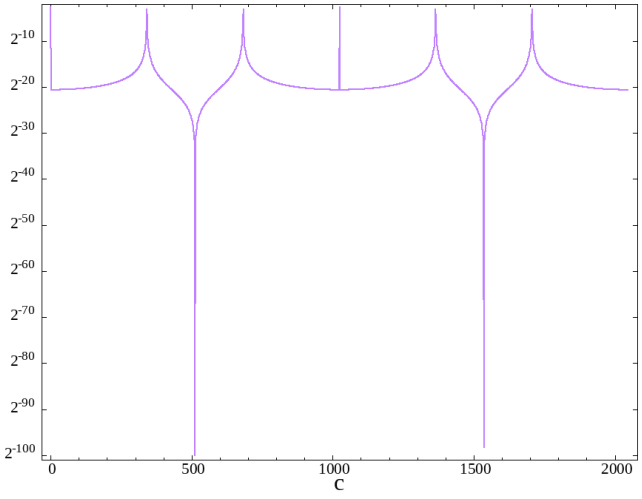
(a) $N = 15 \quad a = 2 \quad t = 5$.



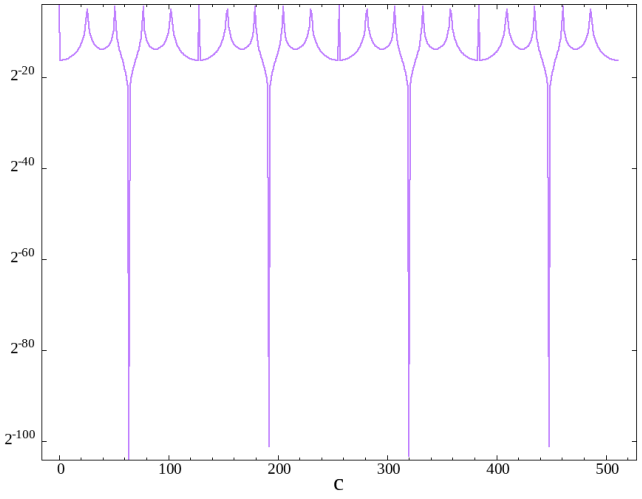
(b) $N = 15 \quad a = 2 \quad t = 9$.



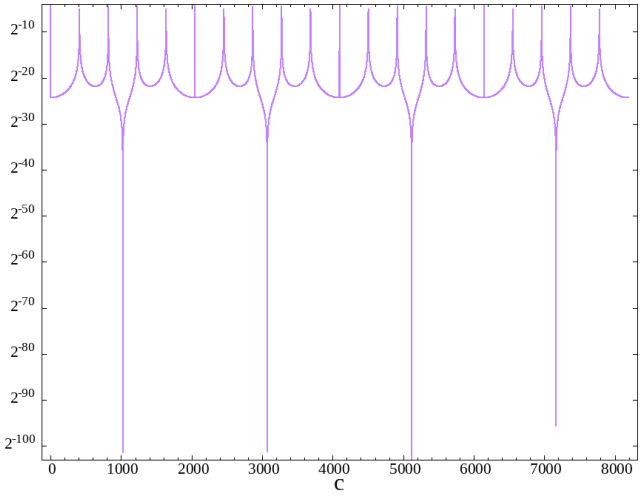
(c) $N = 21 \quad a = 11 \quad t = 7$.



(d) $N = 21 \quad a = 11 \quad t = 11$.

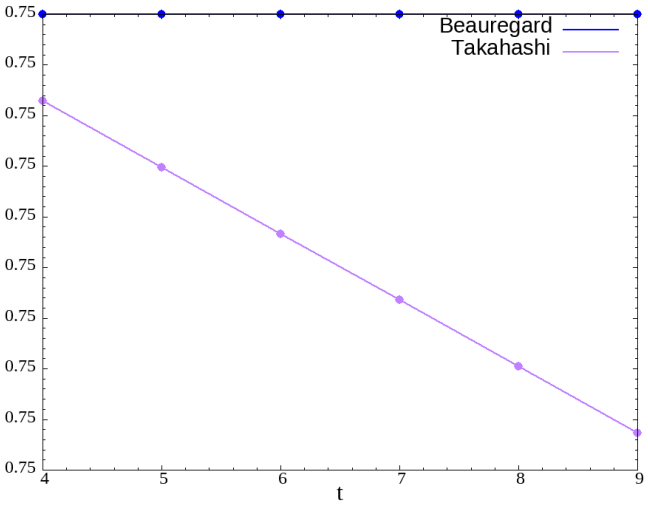


(e) $N = 55 \quad a = 13 \quad t = 9$.

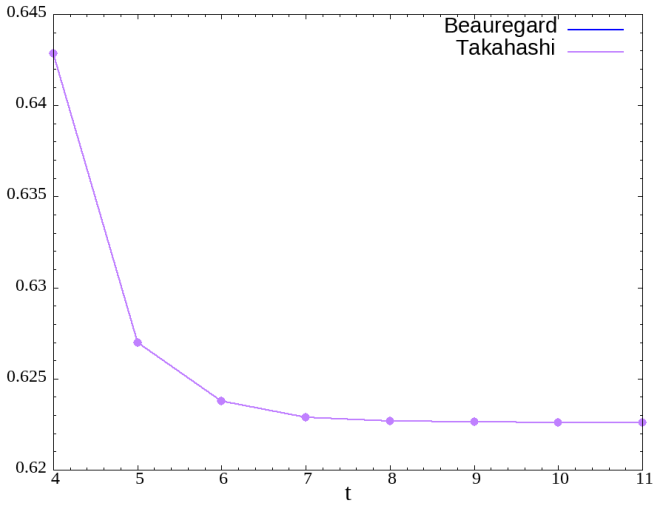


(f) $N = 55 \quad a = 13 \quad t = 13$.

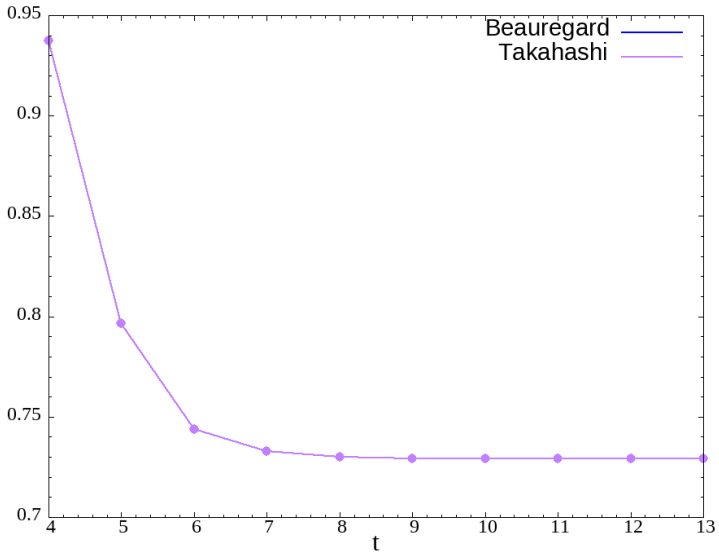
Figura 5.28: Distribución en función de c de los términos de la Divergencia de Kullback-Leibler de las distribuciones de probabilidad obtenidas en las simulaciones con respecto a la distribución de probabilidad teórica del algoritmo de Shor.



(a) $N = 15 \quad a = 2$.



(b) $N = 21 \quad a = 11$.



(c) $N = 55 \quad a = 13$.

Figura 5.29: Probabilidad de medir $c = [qj/r]$ con $j \neq 0$, en función del parámetro de exactitud en la medición de la fase t.

$c = [2^{13}/20] = 410$ es $\sim 2,86\,\%$, pero la probabilidad de medir $c = 409$ es $\sim 1,27\,\%$. Más adelante estudiaremos cuán necesario es que este redondeo correcto se produzca.

Como vimos en 4.3.4, para hallar el $j/r \sim \varphi_j$ se utiliza el algoritmo de fracciones continuas a partir de c/q. A continuación usaremos valores reales obtenidos de nuestros experimentos.

Por ejemplo, empezando por $N = 15$, $a = 2$ y $t = 9$, tenemos que como aquí $r = 4$ y $q/r = 2^9/4 = 2^7 = 128$, entonces no hace falta redondear y las probabilidades se acercan más a $1/r$. En particular, supongamos medimos dos valores $c = 128 \iff j = 1$ y $c = 384 \iff j = 3$, ambos con una probabilidad $\sim 25\,\%$. Entonces, usando fracciones continuas

$$c/q = 128/512$$

1.
$$x = 0,25 \implies i = 0 \implies a_0 = 0$$

2.
$$f = x - i = 0,25 - 0 = 0,25$$

3.
$$x = 1/f = 1/0,25 = 4$$

4.
$$x = 4 \implies i = 4 \implies a_1 = 4 \implies j_1/r_1 = 1/4 \implies r = 4$$

FIN

$$c/q = 384/512$$

1.
$$x = 0,75 \implies i = 0 \implies a_0 = 0$$

2.
$$x = x - i = 0,75 - 0 = 0,75$$

3.
$$x = 1/f = 1/1,3 = 1,3$$

4.

$$x = 1,3 \implies i = 1 \implies a_1 = 1 \implies j_1/r_1 = 1/1 \implies r = 1$$

CONTINÚO

5.

$$x = x - i = 1,3 - 1 = 0,3$$

6.

$$x = 1/f = 1/0,3 = 3$$

7.

$$x = 3 \implies i = 3 \implies a_2 = 3 \implies j_2/r_2 = 1/(1 + 1/3) = 3/4 \implies r = 4$$

FIN

Como no hay redondeo, el algoritmo de fracciones continuas nos da $r = 4$ justo en el último paso de la expansión, cosas que veremos no se da cuando hay redondeo. Ahora, usando $r = 4$ los posibles valores de $p, q \mid N = pq$ son $\gcd(a^{r/2} \pm 1, N)$ por lo que $p = \gcd(2^2 - 1, N) = \gcd(3, 15) = 3$ y $q = \gcd(2^2 + 1, N) = \gcd(5, 15) = 5$.

Pasando a casos más interesantes, ahora veamos $N = 21$, $a = 11$ y $t = 11$, tenemos que aquí $r = 6$ y $q/r = 2^1 1/6$ no es entero, por lo que qj/r da entero sólo si elegimos $j = 3$. En caso contrario, supongamos entonces que medimos los dos valores por encima y por debajo de qj/r con $j = 2$ es $qj/r = 682,6$, o sea, $c = 682$ y $c = 683$, con una probabilidad de $\sim 11,4\,\%$ y $\sim 2,85\,\%$ respectivamente. Entonces, usando fracciones continuas

$$c/q = 682/2048$$

1.

$$x = 682/2048 \implies i = 0 \implies a_0 = 0$$

2.

$$x = x - i = 682/2048 - 0 = 682/2048$$

3.

$$x = 1/f = 2048/682 \sim 3,003$$

4.

$$x = 2048/682 \implies i = 3 \implies a_1 = 3 \implies j_1/r_1 = 1/3 \implies r = 3$$

SIGO

5.

$$x = x - i = 2048/682 - 3 = 2/682$$

6.

$$x = 1/f = 682/2 = 341$$

7.

$$x = 341 \implies i = 341 \implies a_2 = 341$$

$$\implies j_2/r_2 = 1/(1 + 1/341) = 341/342 \implies r = 342$$

ME PASO DE $N = 21$

$$c/q = 683/2048$$

1.

$$x = 683/2048 \implies i = 0 \implies a_0 = 0$$

2.

$$x = x - i = 683/2048 - 0 = 683/2048$$

3.

$$x = 1/f = 2048/683 \sim 2,998$$

4.

$$x = 2048/683 \implies i = 2 \implies a_1 = 2 \implies j_1/r_1 = 1/2 \implies r = 2$$

SIGO

5.

$$x = x - i = 2048/683 - 2 = 682/683$$

6.
$$x = 1/f = 683/682$$

7.
$$x = 683/682 \implies i = 1 \implies a_2 = 1 \implies j_2/r_2 = 1/(2+1/1) = 1/3 \implies r = 3$$

SIGO

8.
$$x = x - i = 683/682 - 1 = 1/683$$

9.
$$x = 1/f = 683$$

10.
$$x = 683 \implies i = 683 \implies a_3 = 683$$
$$\implies j_2/r_2 = \frac{1}{2 + \frac{1}{1+\frac{1}{683}}} = 683/2050 \implies r = 2050$$

ME PASO DE $N = 21$

En los casos en los que hay redondeo, el algoritmo de fracciones continuas no nos da r justo, y llega un punto en que los candidatos r_i para r exceden N. Lo que se hace entonces es probar con los múltiplos del último r_i antes de que excedan N, en este caso 3. Vemos que simplemente multiplicándolo por 2 obtenemos $r = 6$. Los valores de p, q son $p = \gcd(2^3 - 1, N) = \gcd(7, 21) = 7$ y $q = \gcd(2^3 + 1, N) = \gcd(9, 21) = 3$.

Por último $N = 55$ y $a = 13$, En este caso, vamos a mostrar un t no tan grande, en particular $t = 8$, para ver que consecuencias tiene hacer esto. Con $j = 18$, entonces $qj/r = 230, 4$. Voy a ver el caso de medir un valor de c que no es inmediatamente superior o inferior a este número, $c = 229$ que tiene un probabilidad del $\sim 0, 24\,\%$. También, lo más probable para este j con $\sim 2, 87\,\%$ de probabilidad, $c = 230$.

$$c/q = 229/256$$

1.
$$x = 229/256 \implies i = 0 \implies a_0 = 0$$

2.
$$x = x - i = 229/256 - 0 = 229/256$$

3.
$$x = 1/f = 256/229$$

4.
$$x = 256/229 \implies i = 1 \implies a_1 = 1 \implies j_1/r_1 = 1 \implies r = 1$$

SIGO

5.
$$x = x - i = 256/229 - 1 = 27/229$$

6.
$$x = 1/f = 229/27 \sim 8,48$$

7.
$$x = 229/27 \implies i = 8 \implies a_2 = 8 \implies j_2/r_2 = 1/(1+1/8) = 8/9 \implies r = 9$$

SIGO

8.
$$x = x - i = 229/27 - 8 = 13/27$$

9.
$$x = 1/f = 27/13$$

10.
$$x = 27/13 \implies i = 2 \implies a_3 = 2$$
$$\implies j_3/r_3 = \frac{1}{1 + \frac{1}{8+1/2}} = 17/19 \implies r = 19$$

TERMINO

$$c/q = 230/256$$

1.

$$x = 230/256 \implies i = 0 \implies a_0 = 0$$

2.

$$x = x - i = 230/256 - 0 = 230/256$$

3.

$$x = 1/f = 256/230$$

4.

$$x = 256/230 \implies i = 1 \implies a_1 = 1 \implies j_1/r_1 = 1 \implies r = 1$$

SIGO

5.

$$x = x - i = 256/230 - 1 = 13/115$$

6.

$$x = 1/f = 115/13 \sim 8,85$$

7.

$$x = 115/13 \implies i = 8 \implies a_2 = 8 \implies j_2/r_2 = 1/(1+1/8) = 8/9 \implies r = 9$$

SIGO

8.

$$x = x - i = 115/13 - 8 = 11/13$$

9.

$$x = 1/f = 13/11$$

10.

$$x = 13/11 \implies i = 1 \implies a_3 = 1 \implies j_3/r_3 = 1/(1+1/9) = 9/10 \implies r = 10$$

SIGO

11.

$$x = x - i = 13/11 - 1 = 2/11$$

12.

$$x = 1/f = 11/2$$

13.

$$x = 11/2 \implies i = 5 \implies a_4 = 5$$
$$\implies j_3/r_3 = \frac{1}{1 + \frac{1}{8 + \frac{1}{1 + \frac{1}{5}}}} = \frac{53}{59} \implies r = 59$$

ME PASO

Vemos que $c = 230$ nos permite obtener $r = 20$ mucho más simplemente multiplicando el resultado por 2, mientras que $c = 229$ que nos da 19 el cuál no tiene factores comunes con 20, multiplicando obtenemos $r' = 380$. Los valores de p y q, que se obtienen facilmente con $r = 20$, son $p = \gcd((13^{10} - 1) \bmod N, N) = \gcd(33, 55) = 11$ y $q = \gcd((13^{10} + 1) \bmod N, N) = \gcd(35, 55) = 5$.
Por otro lado, con $r' = 380$ son $p = \gcd((13^{190} - 1) \bmod N, N) = \gcd(33, 55) = 11$ y $q = \gcd((13^{190} + 1) \bmod N, N) = \gcd(35, 55) = 5$. Obtenemos el mismo resultado para la factorización, a pesar de no haber encontrado rápidamente r.
Según la figura 5.25, aplicar el algoritmo con $N = 55$, $a = 13$ y $t = 8$ requiere ~ 30000 compuertas, aproximadamente el 60 % que si se hubiera usado $t = 13$. Como vimos en el párrafo anterior, a los fines de calcular los factores de N esto sólo repercutió en un incremento en el número de pruebas con múltiplos de r' y en el cálculo de la exponenciación modular. Resulta aparentemente conveniente reducir casi la mitad las compuertas cuánticas si este es el costo, tanto desde el punto de vista de las simulaciones numéricas como de una implementación física del algoritmo de Shor.

Capítulo 6

Discusión y Conclusiones

En este último capítulo discutiremos los resultados de este trabajo abordándolo desde su conjunto. Se repasarán los aspectos más interesantes del mismo desde el punto de vista de la investigación en el campo de la Computación Cuántica.

6.1. Implementación numérica de compuertas cuánticas

Si bien al momento de la realización de este trabajo existen librerías para la simulación de circuitos como Qiskit [23], la mayoría de estas están basadas en Python, un lenguaje orientado a objetos que no se caracteriza principalmente por su eficiencia, sino por lo intuitivo.
Gracias a que se trabajó con un lenguaje optimizado para el cálculo numérico como lo es FORTRAN, y las posibilidades de paralelización, esta librería es útil para simular sistemas de tamaño considerable, aunque el crecimiento exponencial de los recursos computacionales de simular una computadora clásica es de todas formas muy limitante. Sin embargo, como todas las compuertas y demás operaciones han sido definidas de forma dinámica en función del tamaño del sistema, los resultados obtenidos con n no tan grandes pueden ser igualmente útiles.

Como se vio en la sección 5.3 la precisión numérica de las amplitudes del vector de estado al aplicar compuertas no resulta una limitación, incluso al trabajar números de punto flotante de precisión simple.

6.2. Algoritmos

La implementación desarrollada en este trabajo es especialmente útil a la hora de implementar circuitos extensos, recursivos y con ciclos, como lo es el algoritmo de Shor. Como no se utiliza directamente el modelo de circuitos el cuál no permite ciclos (2.2.2) para llamar a las compuertas

fundamentales que necesita de cada algoritmo, pueden definirse nuevas operaciones mediante ciclos de compuertas fundamentales, dinámicos y escalables en función del tamaño de los sistemas a analizar, y de esta manera implementar eficientemente algoritmos cuánticos.
Una limitación de esta implementación, con respecto a las computadoras cuánticas, es la imposibilidad de reducir la profundidad algorítmica mediante la implementación en paralelo de compuertas fundamentales, es decir, aplicar compuertas sobre varios qubits distintos simultáneamente. Esto se debe a que cada compuerta fundamental actúa sobre y modifica íntegramente el vector de estado de todo el sistema, por lo que deben ejecutarse de forma secuencial. Es por esto que la paralelización sólo se lleva a cabo al nivel de recursividad mas bajo dentro de las compuertas fundamentales, lo cuál contradice las normas no escritas sobre la paralelización de procesos.
Aunque como sabemos simular computadoras cuánticas tiene la obvia desventaja del crecimiento del costo computacional, existe la ventaja de que a diferencia de cuando se trata con sistemas cuánticos reales, las amplitudes de probabilidad son accesibles en cualquier momento de la simulación, incluyendo sus fases. Esto permite corroborar los algoritmos de manera mucho más detallada y a cada paso, ya que incluso las fases absolutas pueden observarse.

6.3. Futuras investigaciones

La librería de compuertas fundamentales que se desarrollo en este trabajo puede ser utilizada para otras implementaciones del algoritmo de Shor, u otros algoritmos completamente distintos como los algoritmos de búsqueda de Grover [24].
Aunque se lo haya citado, no se utilizo lo novedoso del articulo de Häner que es el sumador TOFFOLI [9]. Un futuro trabajo podría incluir implementar este algoritmo, que utiliza $2n + 2$ qubits. Otros autores como C. Gidney [13] y C. Zalka [14] se centraron también en reducir el número de qubits, logrando implementaciones que usan un total de $2n + 1$ y $1,5n + O(1)$ qubits respectivamente.
También, se puede implementar el circuito desarrollado por X. Liu et al. [12], el cuál cambia el enfoque estándar de reducir el número de qubits a focalizarse en reducir el número de compuertas CNOT.
En las implementaciones experimentales de computación cuántica no es posible aplicar rotaciones con fases arbitrarias, por lo que debe llevarse a cabo la *síntesis* de las compuertas de rotación. Esto es, llevar a cabo la rotación en ϕ con una precisión finita utilizando un conjunto de compuertas de rotación con fases fija como Z, S o T. Una sugerencia de trabajo posterior es implementar un algoritmo para llevar a cabo dicha síntesis dentro de la propia librería.
También en relación a las rotaciones, en los circuitos implementados en este trabajo hay varias oportunidades para ahorrar compuertas de rotación, que no se aprovecharon. Por ejemplo, en

las compuertas $\phi ADD(a)$, como ya sabemos a de antemano podemos sumar las fases que de las rotaciones que deben aplicarse sobre cada qubit, y hacerlo usando una sola compuerta de rotación. Es interesante buscar más oportunidades para hacer esto mismo.

Una potencial mejora a nivel fundamental de la librería sería, siguiendo la idea de la Esfera de Bloch (figura 3.1), utilizar la representación polar para almacenar los 2^n coeficientes complejos de los vectores de estado. Esto es, en vez de representar el coeficiente $a_x = \mathrm{Re}(a_x) + i\mathrm{Im}(a_x)$ como se hizo en este trabajo, usar $\rho_x \in [0; 1)$ y $\theta_x \in [0, 2\pi)$. De esta forma estaríamos almacenando directamente cantidades más útiles, siendo $P(x) = \rho_x^2$ la probabilidad de medir el estado x y θ_x la fase que como vimos en la estimación de fase (subsección 4.2.3) es donde se almacena la información útil en muchos algoritmos. Sin embargo, los costos computacionales de calcular $\mathrm{Re}(a_x) = \rho_x \cos(\theta_x)$ y $\mathrm{Im}(a_x) = \rho_x \sin(\theta_x)$ probablemente excedan los beneficios.

En este trabajo no se ha considerado la introducción de errores debido a la decoherencia de los qubits, un fenómeno muy importante en la práctica. Sería interesante investigar como responden las compuertas y los algoritmos ante errores como la aparición de fases relativas aleatorias. En línea con esto, se podrían implementar con la librería códigos cuánticos de corrección de errores e estudiar el impacto de estos.

Desde el punto de vista de la programación y el desarrollo de software, sería útil convertir el código en una librería a derecho propio, que pueda ser descargada, instalada e utilizada para simular algoritmos cuánticos. También una interfaz de usuario más amigable o un nexo a otros lenguajes de programación como Python motivaría el uso de la misma por usuarios menos habituados a la programación en general o al lenguaje FORTRAN.

Apéndice A

Sistema Criptográfico RSA

Como se menciona repetidamente a lo largo del trabajo, el algoritmo de factorización de Shor fue uno de los mayores impulsos a la computación cuántica debido a su potencial capacidad para volver obsoleto el sistema de encriptación más usado en la actualidad, el sistema RSA. El mismo, que debe su nombre a sus creadores, Rivest, Shamir y Adleman [7], es un sistema criptográfico de clave pública. Los sistemas criptográficos de clave pública funcionan de manera general de la siguiente forma: Una persona que desee recibir mensajes seguros, que por convención llamaremos Alice, genera dos claves, una clave *pública* P y una clave *privada* S. Luego, distribuye la clave P a todo el mundo, mientras que almacena S de forma segura. El emisor de un mensaje, Bob, utiliza P para encriptar el mismo y lo envía al receptor quien, utilizando la clave S, desencripta el mensaje para obtener la información. Un tercero, Eva, que desee robar la información contenida en el mensaje que ha interceptado, necesita desencriptar el mismo sin poseer S. La seguridad de cierto sistema de clave pública radica en la dificultad de desencriptar mensajes, que es equivalente a obtener S, a partir de la clave pública P. S se utiliza para realizar la operación inversa a la que usa para encriptar mediante P, por lo que están relacionadas entre sí.
El sistema RSA basa su fortaleza en la dificultad del problema de factorización en computadoras clásicas, como explicaremos a continuación. Supongamos que Alice quiere recibir mensajes encriptados mediante el sistema RSA, por lo que procede de la siguiente forma:

1. Elige dos números primos grandes, p y q.
2. Calcula el producto $N = pq$
3. Elige al azar un entero impar pequeño e, tal que sea coprimo con $\varphi(N) = (p-1)(q-1)$
4. Calcula d, el inverso multiplicativo de e mod $\varphi(N)$, es decir, ed mod $\varphi(N) = 1$
5. El par $P = (e, N)$ y el par $S = (d, N)$ constituyen las claves pública y privada del sistema RSA, respectivamente

Si Bob quiere enviar un mensaje a Alice, utiliza (e, N). Supongamos sin pérdida de generalidad que el mensaje M tiene sólo $\lfloor \log N \rfloor$ bits (mensajes más largos pueden descomponerse en mensajes de este tamaño). El procedimiento de encriptación es el siguiente

$$E(M) = M^e \text{mod } N$$

donde $E(M)$ es la versión encriptada del mensaje que Bob quiere mandar a Alice. Una vez recibido, ella puede desencriptarlo fácilmente con la clave privada (d, N), simplemente elevando el mensaje encriptado a la d-ésima potencia:

$$D(E(M)) = (E(M)^d) \text{mod } N$$

Para que la desencriptación sea exitosa, tenemos que recuperar el mensaje original, o sea $D(E(M)) = M \text{ mod } N$. Esto surge a partir de la propia definición de d como inverso multiplicativo mod $\varphi(N)$, que implica que $ed = 1 + k\varphi(N)$ con k entero. Consideremos ahora dos casos particulares. Si M es coprimo con N, entonces por la generalización del Euler del Pequeño Teorema de Fermat, se sigue que $M^{k\varphi(N)} = 1 \text{ mod } N$ y por lo tanto,

$$\begin{aligned} D(E(M)) &= E(M)^d \text{mod } N \\ &= M^{ed} \text{mod } N \\ &= M^{1+k\varphi(N)} \text{mod } N \\ &= (M(M^{k\varphi(N)})) \text{mod } N \\ &= M \text{ mod } N \end{aligned}$$

por lo que si M es coprimo con N la desencriptación funciona. En caso contrario, se tiene que p o q o bien ambos dividen a M. Sin pérdida de generalidad supongamos que sólo p divide M. Luego, vale que $M = 0 \text{ mod } N$ y por lo tanto $M^{ed} = 0 \text{ mod } N$. Como q no divide a M tenemos que $M^{q-1} = 1 \text{ mod } q$ por el Pequeño Teorema de Fermat, y por consiguiente que $M^{\varphi(N)} = 1 \text{ mod } N$ ya que $\varphi(N) = (p-1)(q-1)$. Usando ahora que $ed = 1 + k\varphi(N)$ vemos que $M^{ed} = M \text{ mod } q$ y gracias al Teorema Chino del resto llegamos a que se debe cumplir que $M^{ed} = M \text{ mod} N$. De esta forma, vemos que en este caso también es exitosa la desencriptación.

Una vez visto como funciona RSA, veamos porque un algoritmo de factorización eficiente para números grandes podría volverlo obsoleto. Supongamos que Eva puede factorizar N en sus factores primos p y q, y que puede computar $\varphi(N) = (p-1)(q-1)$. Con el conocimiento de e, es fácil obtener d, que permite desencriptar el mensaje.
También, es posible romper RSA sin llegar a factorizar N, sino encontrando el orden del mensaje encriptado. Es decir, el menor entero r tal que $(M^e)^r = 1 \text{ mod } N$. Podemos suponer esto, o sea que M^e es coprimo de N, sin perdida de generalidad ya que de lo contrario $M^e \text{ mod } N$ y N tienen

factores comunes que pueden ser extraídos mediante el algoritmo de Euclídes. Puede probarse que r divide a $\varphi(N)$, y como e es coprimo con $\varphi(N)$ tambien lo es con r y por lo tanto tiene un inverso multiplicativo mod r. Sea d' este inverso, tal que $ed' = 1 + kr$ para cierto entero k, entonces Eva puede recobrar el mensaje original M elevando el mensaje encriptado a la d'-ésima potencia.

$$\begin{aligned}(M^e)^{d'} &= M^{1+kr} \text{mod } N \\ &= (M(M^{kr})) \text{mod } N \\ &= M \text{ mod } N\end{aligned}$$

En este caso, podemos ver que es posible desencriptar el mensaje sin llegar nunca a conocer exactamente la clave privada $S = (d, N)$, sino sólo (d', N). Aunque este método no requiera factorizar N, el problema de hallar el orden es un paso fundamental en la factorización, y no existe algoritmo clásico que lo resuelva eficientemente. Es justamente en la dificultad del problema de hallar el orden en lo que se basa la seguridad del sistema RSA. Sin embargo, este sistema podría ser susceptible a ser atacado de otras maneras, aunque a la fecha no se han encontrado formas de hacerlo.

Bibliografía

[1] Peter W Shor. «Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer». En: *SIAM review* 41.2 (1999), págs. 303-332.

[2] Yasuhiro Takahashi y Noboru Kunihiro. «A quantum circuit for Shor's factoring algorithm using 2n+ 2 qubits». En: *Quantum Information & Computation* 6.2 (2006), págs. 184-192.

[3] Stephane Beauregard. «Circuit for Shor's algorithm using 2n+ 3 qubits». En: *arXiv preprint quant-ph/0205095* (2002).

[4] Richard P Feynman y col. «Simulating physics with computers». En: *Int. j. Theor. phys* 21.6/7 (2018).

[5] Yu I Manin. *Vychislimoe i nevychislimoe (Computable and Noncomputable), Moscow: Sov.* 1980.

[6] Peter W Shor. «Algorithms for quantum computation: discrete logarithms and factoring». En: *Proceedings 35th annual symposium on foundations of computer science*. Ieee. 1994, págs. 124-134.

[7] Ronald L Rivest, Adi Shamir y Leonard Adleman. «A method for obtaining digital signatures and public-key cryptosystems». En: *Communications of the ACM* 21.2 (1978), págs. 120-126.

[8] David P DiVincenzo. «The physical implementation of quantum computation». En: *Fortschritte der Physik: Progress of Physics* 48.9-11 (2000), págs. 771-783.

[9] Thomas Häner, Martin Roetteler y Krysta M Svore. «Factoring using 2n+ 2 qubits with Toffoli based modular multiplication». En: *arXiv preprint arXiv:1611.07995* (2016).

[10] Vlatko Vedral, Adriano Barenco y Artur Ekert. «Quantum networks for elementary arithmetic operations». En: *Physical Review A* 54.1 (1996), pág. 147.

[11] David Beckman y col. «Efficient networks for quantum factoring». En: *Physical Review A* 54.2 (1996), págs. 1034-1063. DOI: 10.1103/physreva.54.1034. URL: https://doi.org/10.1103%2Fphysreva.54.1034.

[12] Xia Liu, Huan Yang y Li Yang. *CNOT-count optimized quantum circuit of the Shor's algorithm*. 2021. arXiv: 2112.11358 [quant-ph].

[13] Craig Gidney. *Factoring with n+2 clean qubits and n-1 dirty qubits*. 2018. arXiv: 1706.07884 [quant-ph].

[14] Christof Zalka. *Shor's algorithm with fewer (pure) qubits*. 2006. arXiv: quant-ph/0601097 [quant-ph].

[15] Michael A Nielsen e Isaac Chuang. *Quantum computation and quantum information*. American Association of Physics Teachers, 2002.

[16] Adriano Barenco y col. «Elementary gates for quantum computation». En: *Physical review A* 52.5 (1995), pág. 3457.

[17] S Parker y Martin B Plenio. «Efficient factorization with a single pure qubit and log N mixed qubits». En: *Physical Review Letters* 85.14 (2000), pág. 3049.

[18] Thomas G Draper. «Addition on a quantum computer». En: *arXiv preprint quant-ph/0008033* (2000).

[19] Robert B Griffiths y Chi-Sheng Niu. «Semiclassical Fourier transform for quantum computation». En: *Physical Review Letters* 76.17 (1996), pág. 3228.

[20] William H Press y col. *Numerical recipes 3rd edition: The art of scientific computing*. Cambridge university press, 2007.

[21] Rohit Chandra. *Parallel programming in OpenMP*. Morgan kaufmann, 2001.

[22] S. Kullback y R. A. Leibler. «On Information and Sufficiency». En: *The Annals of Mathematical Statistics* 22.1 (1951), págs. 79-86. DOI: 10.1214/aoms/1177729694. URL: https://doi.org/10.1214/aoms/1177729694.

[23] Qiskit contributors. *Qiskit: An Open-source Framework for Quantum Computing*. 2023. DOI: 10.5281/zenodo.2573505.

[24] Lov K. Grover. «A fast quantum mechanical algorithm for database search». En: *arXiv e-prints*, quant-ph/9605043 (mayo de 1996), quant-ph/9605043. DOI: 10.48550/arXiv.quant-ph/9605043. arXiv: quant-ph/9605043 [quant-ph].

[25] Jos Thijssen. *Computational physics*. Cambridge university press, 2007.

[26] Noson S Yanofsky y Mirco A Mannucci. *Quantum computing for computer scientists*. Cambridge University Press, 2008.

[27] Pieter Kok y Brendon W Lovett. *Introduction to optical quantum information processing*. Cambridge university press, 2010.

[28] Michel Le Bellac. *A short introduction to quantum information and quantum computation*. Cambridge University Press, 2006.

[29] Tudor D Stanescu. *Introduction to topological quantum matter & quantum computation*. CRC Press, 2016.

[30] Susanna F Huelga y col. «Improvement of frequency standards with quantum entanglement». En: *Physical Review Letters* 79.20 (1997), pág. 3865.

[31] Richard Cleve. «An introduction to quantum complexity theory». En: *Collected Papers on Quantum Computation and Quantum Information Theory* (2000), págs. 103-127.

[32] G Eric Moorhouse y UW Math. *Shor's algorithm for factorizing large integers*. 1998.

[33] Christof Zalka. «Fast versions of Shor's quantum factoring algorithm». En: *arXiv preprint quant-ph/9806084* (1998).

[34] Steven A Cuccaro y col. «A new quantum ripple-carry addition circuit». En: *arXiv preprint quant-ph/0410184* (2004).

[35] Igor L Markov y Mehdi Saeedi. «Constant-optimized quantum circuits for modular multiplication and exponentiation». En: *arXiv preprint arXiv:1202.6614* (2012).

[36] Rodney Van Meter y Kohei M Itoh. «Fast quantum modular exponentiation». En: *Physical Review A* 71.5 (2005), pág. 052320.

[37] R. Cleve y col. «Quantum algorithms revisited». En: *Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences* 454.1969 (ene. de 1998), págs. 339-354. DOI: 10.1098/rspa.1998.0164. URL: https://doi.org/10.1098/Frspa.1998.0164.

[38] Wikimedia Commons. *File:Quantum phase estimation.svg — Wikimedia Commons, the free media repository*. [Online February-2023]. 2020. URL: %5Curl%7Bhttps://commons.wikimedia.org/w/index.php?title=File:Quantum_phase_estimation.svg&oldid=493292837%7D.

[39] Wikimedia Commons. *File:Shor's algorithm.svg — Wikimedia Commons, the free media repository*. [Online February-2023]. 2020. URL: %5Curl%7Bhttps://commons.wikimedia.org/w/index.php?title=File:Shor%27s_algorithm.svg&oldid=492679607%7D.

[40] Michael Sipser. *Introduction to the Theory of Computation*. Vol. 27. 1. ACM New York, NY, USA, 1996, págs. 27-29.

[41] Sanjeev Arora y Boaz Barak. *Computational complexity: a modern approach*. Cambridge University Press, 2009.