Centro de Investigación Científica y de Educación Superior de Ensenada, Baja California

Maestría en Ciencias
en Ciencias de la Computación

Jumbled matching cuántico

Tesis

para cubrir parcialmente los requisitos necesarios para obtener el grado de

Maestro en Ciencias

Presenta:

Julio César Juárez Xochitemol

Ensenada, Baja California, México

2021

Tesis defendida por

Julio César Juárez Xochitemol

y aprobada por el siguiente Comité

Dr. Edgar Leonel Chávez González

Director de tesis

Dr. José Alberto Fernández Zepeda

Dr. Pedro Gilberto López Mariscal

Dr. Raúl Rangel Rojo

Dr. Pedro Gilberto López Marisca

Coordinador del Posgrado en Ciencias de la Computación

Dr. Pedro Negrete Regagnon

Director de Estudios de Posgrado

Resumen de la tesis que presenta Julio César Juárez Xochitemol como requisito parcial para la obtención del grado de Maestro en Ciencias en Ciencias de la Computación.

Jumbled matching cuántico

Resumen aprobado por:

Dr. Edgar Leonel Chávez González
Director de tesis

La computación cuántica nace de la observación de Richard Feynman de la dificultad de simular sistemas cuánticos en la máquina RAM. El interés por este modelo de cómputo llegó con el algoritmo de Shor, donde la factorización de números, un problema en la intersección entre NP y co-NP, y del cual dependen muchos sistemas criptográficos, se pudo hacer en tiempo polinomial. Otra sorpresa fue el algoritmo de Grover, que permite encontrar un objeto en una colección en menos pasos que el tamaño de la colección; contrario incluso a la intuición natural. Con esta motivación, se aborda un problema que también requiere una cantidad lineal de pasos; pero en donde la condición de paro es mas compleja que la condición de igualdad. En este trabajo se estudia el problema de *jumbled pattern matching (JPM)*, donde dada una cadena w y un patrón p tales que $w, p \in \Sigma^*$ y $|p| \leq |w|$, buscamos los sub-patrones de w que estén construidos con los mismos elementos que p; es decir, todos los sub-patrones que sean una permutación de p. El JPM es especialmente útil en problemas de biocomputación relacionados con análisis de patrones, como espectrometría de masa para interpretación de datos, descubrimientos de SNP, alineamiento de secuencias, y aplicaciones de biosecuenciación. Explorando la computación cuántica, se presenta un algoritmo de complejidad $O(\sqrt{N})$ que genera la solución al problema de *jumbled pattern matching* casi seguramente; por tanto, acelerando las mejores soluciones clásicas en un factor cuadrático.

Palabras clave: computación cuántica, algoritmo de Grover, emparejamiento entremezclado, aceleración cuadrática

Abstract of the thesis presented by Julio César Juárez Xochitemol as a partial requirement to obtain the Master of Science degree in Computer Science.

Quantum jumbled matching

Abstract approved by:

Dr. Edgar Leonel Chávez González
Thesis Director

Quantum computing was born from Richard Feynman's observation of the difficulty of simulating quantum systems in the RAM machine. The interest in this computational model arose with the Shor's algorithm, where the integer factorization, a problem in the intersection between NP and co-NP, on which many cryptography systems depend, it can be performed in polynomial time. Another surprise was the Grover's algorithm, which allows finding an object among a collection in fewer steps than the size of the collection; contrary even to natural intuition. With this motivation we work on a problem that also requires a linear number of steps; but where the halting condition is more complex than the equality condition. In this work we approach the *jumbled pattern matching* problem, where given a text w and a pattern p such that $w, p \in \Sigma^*$ and $|p| \leq |w|$, we search for the sub-patterns in w built with the same elements of p; that is, all sub-patterns that are a permutation of p. JPM is specially useful in bio computing problems related to pattern analysis, as mass spectrometry for data interpretation, SNP discovery, sequence alignment, and bio sequencing applications. Exploring quantum computing, we present an algorithm whose complexity is $O(\sqrt{N})$ that generates the solution for *jumbled pattern matching* almost certainly; therefore, speeding-up the best classical solutions in a quadratic factor.

Keywords: quantum computing, Grover's algorithm, jumbled matching, quadratic speed-up

Dedicatoria

En memoria de mi padre.

Agradecimientos

A mis padres Julio y Mercedes por su apoyo incondicional, su amor y paciencia. Siempre están en mi mente, y por ustedes es que todo tiene sentido, incluyendo esta tesis de maestría.

A mis hermanos Mónica y Ubaldo por todas las risas, por ser junto con mamá, lo más valioso que tengo en este mundo.

A Dianita y Nohemi por darme su apoyo incondicional todos estos años, por ser las mejores amigas que podría encontrar en esta vida.

Al Dr. Edgar Chávez por esas clases de fundamentos de computación tan fantásticas, por creer en mi y permitirme hacer el trabajo de tesis con usted.

A mis sinodales por toda la retroalimentación que me han dado a lo largo de estos meses de trabajo.

Al Centro de Investigación Científica y de Educación Superior de Ensenada.

Al Consejo Nacional de Ciencia y Tecnología (CONACyT) por brindarme el apoyo económico para realizar mis estudios de maestría.

Tabla de contenido

Página

Tabla de contenido (continuación)

Lista de figuras

Figura Página

Lista de figuras (continuación)

Lista de tablas

Capítulo 1. Introducción

1.1. Antecedentes

Cuando Turing, Gödel y Church publicaron los primeros modelos de cómputo tal vez nunca imaginaron lo revolucionarias que serían aquellas ideas. El modelo de Turing utilizando un cabezal con la capacidad de imprimir símbolos sobre una banda infinita (Turing, 1937), el de Gödel usando funciones recursivas que podían generar cosas como la secuencia de Fibonnaci (Gödel, 1931), y finalmente el de Church siendo un método de definición de funciones como reglas de cómputo (Church, 1936). Estos modelos que parecían ser tan distintos, resultaron ser equivalentes entre ellos, llegando así a la *Máquina de Turing Determinista (MTD)*, el modelo que más influencia ha tenido en la teoría de la computación. Esta máquina es una *máquina de estados finitos* donde a partir de un estado inicial la máquina evoluciona bajo reglas bien definidas, cada vez que avanza a un siguiente estado realiza una acción sobre la banda y una vez que se ha alcanzado el estado final (o estado de aceptación) la máquina se detiene y genera una salida; se le denomina determinista porque cada vez que se realizan acciones sólo existe una manera de ejecutar la siguiente acción dada la configuración actual de la máquina.

Con el tiempo nacieron otros modelos de computación, llegando a las *Máquinas de Acceso Aleatorio* o *RAM*, un modelo que generalmente utilizan los computólogos para analizar algoritmos y probar su tiempo de ejecución, o dicho de otro modo, determinar la cantidad de pasos que toma a la máquina ejecutar el algoritmo (Cook y Reckhow, 1972). Este modelo consta de un conjunto básico de operaciones y un registro de memoria al que puede acceder en tiempo constante (se puede decir que instantáneamente) y que parecería ser más potente que el modelo ideado por Turing. Sin embargo, cualquier cálculo que pueda ejecutar el modelo *RAM* se puede ejecutar con las máquinas de Turing, aunque a costa de tardarse un tiempo polinomialmente mayor. Este modelo es cercano al que realmente se basa casi toda la infrastructura computacional, la *Arquitectura Von Neumann*.

Ha pasado más de un siglo desde las ideas teóricas detrás de las computadoras

actuales, y muchas décadas desde las primeras computadoras hechas a partir de tubos de vacío que consumían ingentes cantidades de energía, que comparándolas con nuestros teléfonos inteligentes son varios órdenes de magnitud menos potentes. Además el hardware ha llegado a niveles de miniaturización que rozan la ciencia ficción con tecnologías que fabrican transistores con sólo unos pocos átomos. A pesar de todo, ni esa inmensa cantidad de transistores y su fuerza bruta puede hacer mejoras en muchos de nuestros algoritmos debido a razones más fundamentales que nada tienen que ver con la arquitectura o con la manera en la que se fabrican las computadoras. Aún con estas densidades extremas de transistores tradicionales, los nuevos prototipos diseñados a partir de circuitos ópticos con velocidades de conmutación a velocidad de la luz, computación construida con sistemas biológicos o cualquier otra forma novedosa; mediante conceptos como la dureza condicional y reducciones se puede probar que existen algoritmos cuyo tiempo de ejecución no se puede mejorar al usar computadoras clásicas. Dicho en pocas palabras,son equivalentes a las *Máquinas de Turing Deterministas* (MTD) sobre las cuales se hacen esas pruebas de optimalidad.

En la teoría de la computación existe un tipo de máquina de cómputo que podría acelerar esos algoritmos probados óptimos en las MTD, a estas máquinas se les llama *Máquinas de Turing No Deterministas (MTND)*. A diferencia de las MTD donde dada una configuración de la máquina sólo hay una posible acción a ejecutar, en las MTND existen varias acciones posibles; y de entre todas estas, como si fuera un apostador perfecto, la máquina *adivina* cuál es la opción que conducirá a un estado de aceptación. Se podría decir que las MTND pueden explorar exhaustivamente todos los posibles caminos computacionales y escoger el más apropiado. El gran problema de las MTND es que nadie tiene idea de cómo se podrían construir estas máquinas que consiguen resolver en tiempo polinomial problemas de los cuales se ha conjeturado la no existencia de algoritmos que los resuelvan en tiempo subexponencial. Un ejemplo de estos problemas es $k-$SAT, donde dada una fórmula normal conjuntiva (una expresión que consta de conjunciones de disyunciones), se desea saber si existe una asignación de variables tal que la fórmula es verdadera. A la conjetura de la imposibilidad de resolver $k-$SAT en tiempo subexponencial se le conoce como *Hipótesis Fuerte de Tiempo Exponencial (SETH)*.

Pero no todo está perdido, aunque por ahora las MTND son imposibles de fabricar, se tiene una alternativa a medio camino entre las MTD y las MTND. Desde la época de Feynman se imaginaron alternativas que hacían uso de estados de sistemas cuánticos que en teoría acelerarían algoritmos considerados y probados como los mejores dentro del paradigma de la computación clásica. Lo anterior, que puede parecer magia, es posible gracias a que los sistemas cuánticos tienen una característica muy interesante; pueden mantener en superposición una cantidad exponencial de posibles estados con tan sólo una cantidad lineal de memoria. Esto no quiere decir que se haga una representación explícita de dichos estados, de ser así se tendría que mantener en memoría cada uno de los parámetros de los estados, utilizando una cantidad exponencial de memoria. Una forma de ver esta característica es que esos estados están de algún modo *viviendo* al mismo tiempo. En el caso de las MTND cuando se les pide una solución, esta se genera siempre que exista el estado de aceptación, mientras que en el sistema cuántico el estado que se genera es uno de los tantos que se encuentran en superposición; y por supuesto, no siempre será un estado de aceptación, sino algún estado dada cierta distribución de probabilidad. Los sistemas cuánticos que se usan con la finalidad de realizar computación se definen como *registros de memoria cuánticos* y se forman por análogos cuánticos de los bits llamados *qubits* o *quantum-bits*.

Usar sistemas computacionales cuánticos ha sido y parece que seguirá siendo una tarea complicada, principalmente por la delicadez que requiere el manejo de los qubits, ya que cualquier perturbación tiene la capacidad de introducir errores. Errores como la aplicación de compuertas (los objetos que hacen evolucionar el sistema cuántico) incorrectas; errores por desconocimiento parcial de la dinámica del sistema cuántico a implementar. Esto es, que el sistema que se supone es gobernado por un Hamiltoniano H, en realidad esté gobernado por otro Hamiltoniano H' ligeramente distinto. Hasta errores inherentes al entorno del que se desea aislar el sistema, dado que en la práctica el aislamiento perfecto es imposible. En la teoría de los algoritmos cuánticos no es de interés tomar en cuenta los errores que induce el entorno al registro de memoria, pero la realidad es que para obtener *qubits lógicos* útiles para realizar cómputo cuántico, se requieren de muchos más *qubits físicos*. Hay que tener en cuenta que aún cuando se logre un vacío “perfecto” en el cual poner el sistema; en este vacío existen fluctuaciones cuánticas que pueden interactuar con los qubits, alterando su estado y

por tanto causar errores.

Ampliando sobre los errores que ocasiona la inexactitud de los Hamiltonianos involucrados en la dinámica, queda mucho camino por recorrer. En primer lugar, porque el diseño de las computadoras cuánticas se encuentra en una etapa parecida a la de los años cincuenta en la computación clásica, sin una arquitectura definida, donde cada una de las compuertas que evolucionarán el sistema se implementa por un cierto elemento de algún sistema físico; y al haber tantos tipos distintos de sistemas cuánticos para su construcción es muy complicado saber cuál es el Hamiltoniano preciso que representa la evolución del sistema. Por mencionar algunos de los sistemas físicos que utilizan los científicos experimentales se encuentran los sistemas ópticos lineales, trampas de iones, superconductores, resonancia magnética nuclear entre muchos otros. Con lo que en este momento del desarrollo de la computación cuántica es imposible saber cuál de esos sistemas será el más adecuado para la implementación final a gran escala o si algún día se logrará esa implementación. Por lo anterior, no existen máquinas virtuales donde se puedan emular otras computadoras cuánticas; cada una al tener hardware único, su codificación requiere de software único desarrollado por los creadores de cada máquina.

Obtener un registro de memoria adecuado requiere de técnicas de detección y corrección de errores, de precisión en la dinámica de los sistemas y de muchos otros parámetros ya discutidos en el párrafo anterior. No obstante, si los físicos e ingenieros logran lidiar con estos problemas se podría cambiar radicalmente la forma de hacer cálculos al ser capaces de implementar *entrelazamiento*, *superposición*, *teleportación* y toda la artillería que la mecánica cuántica ofrece. Aunque no se sabe si es posible la construcción a gran escala de las computadoras cuánticas y que cientos de problemas teóricos y técnicos se siguen investigando; la construcción de algoritmos debe seguir avanzando, desarrollarlos pensando en que algún día se podrán ejecutar es parte del propio avance de la computación cuántica.

1.2. El problema de jumbled matching

Las cadenas de caracteres juegan un papel importante en biología computacional, recuperación de la información, espectrometría y otras áreas. Se puede cuantificar la similaridad entre cadenas mediante distintas métricas; por ejemplo, si las cadenas son de distinto tamaño se puede usar la distancia de edición o Levenshtein, que es la cantidad de inserciones, borrados y sustituciones que se deben aplicar a una cadena para convertirla en otra. Si son del mismo tamaño podemos aplicar la distancia de Hamming, que mide el número de posiciones en las que una cadena es distinta a otra. Existe una tercera forma de comparar la similaridad, a la que llamaremos distancia de *revoltura* o *jumbled*, aunque en realidad más que una *distancia* es una semimétrica discreta a la que denotamos como *matching*. Se dice que dos cadenas hacen *jumbled match* si están construidas con los mismos caracteres y la misma cantidad de estos. La salida de esta comparación es 0 o 1, donde la *distancia* 0 es un match, no será un match en caso contrario. Entonces se puede definir al problema de *emparejamiento entremezclado* o de *jumbled matching* como la búsqueda en un texto de todas las cadenas que hacen *jumbled match* con una cadena consulta.

1.2.1. Solución al problema

Para resolver este problema se puede utilizar índices, donde se tienen miles de cadenas de un lado y una consulta por el otro. En el caso de las distancias de Hamming o de edición se desea encontrar aquellas cadenas con la distancia mínima a la consulta, mientras que en matching se desea encontrar todas aquellas cadenas que coincidan con la consulta (distancia 0). La alternativa secuencial es, en ambos casos, comparar la consulta con cada una de las cadenas y quedarse con la de distancia menor, o aquellas que coincidan para el caso de la semi métrica discreta *jumbled match* que definimos an la subsección anterior.

En la búsqueda indexada, se construye un índice previo a las consultas, para después buscar las coincidencias en tiempo constante. Éste problema ha sido muy difícil de estudiar, se ha intentado mejorar la solución por más de 40 años al buscar un índice universal; sin embargo, no ha sido posible hasta ahora. Un caso particular consiste

en tener un sólo texto en el que se van a considerar todas las subcadenas de tamaño m, donde m es el tamaño de la consulta, y se desea encontrar cuales emparejan a la consulta. Hay dos soluciones triviales:

- *Sin índice*, en tiempo lineal con un algoritmo de búsqueda por ventana deslizante.
- *Con índice*, usando espacio y tiempo cuadrático para construir el índice de todas las subcadenas; es decir, construir diccionarios para tamaños $2, 3, 4, .., n$, cada diccionario representa un distinto *vector de Parikh* y tiempo constante de consulta.

Sobre este último método no se sabía si era posible construir índices en menos de $O(n^2)$; sin embargo, se probó en 2014 usando dureza condicional, que no es posible. Este resultado de dureza no depende de algún algoritmo específico (Amir *et al.* (2014)). Dar solución a este problema implica encontrar subrutinas en algoritmos de espectrometría de masa, análisis de similaridad entre secuencias de proteínas y hasta descubrimientos de polimorfismos en ADN.

El resultado anterior implica un límite duro para el problema, la pregunta natural es ¿Hay algo más que se pueda hacer? El resultado de Amir *et al.* (2014) implica que al menos en computación clásica parece que no; sin embargo, en la computación cuántica existe una alternativa que podría acelerar la solución clásica, y todo gracias a un algoritmo llamado *Algoritmo de Grover* (que se verá a profundidad en secciones posteriores) diseñado por Lov Kumar Grover en 1996 (Grover, 1996). El problema que resuelve el algoritmo de Grover es saber si un cierto elemento está, o no, en un conjunto; las computadoras clásicas no pueden resolverlo en un tiempo menor al lineal pues si el conjunto sobre el que se hará la búsqueda no está estructurado lo mejor que se puede hacer es revisar todo el conjunto hasta localizar el elemento deseado; por lo que en tiempo promedio se tardaría $N/2$ pasos y en el peor de los casos se visitaría todos los elementos, tardando N pasos. El algoritmo de Grover localiza el elemento en el arreglo en $O(\sqrt{N})$ pasos o tiempo sublineal, por tanto, con base en este algoritmo y sus generalizaciones, se explorará la posibilidad de construir un algoritmo cuántico para el problema de jumbled matching con un mejor tiempo de ejecución.

1.3. Hipótesis

Cualquier función que pueda calcular una computadora clásica también la puede calcular una computadora cuántica. En otras palabras, para cada algoritmo clásico existe su análogo en la computación cuántica; sin embargo, aún no se ha descubierto una forma precisa de realizar el traslado de un algoritmo clásico cualquiera a su correspondiente cuántico. Pero como se ha visto en Mateus y Omar (2005) y Hariharan y Vinay (2003) existen algoritmos cuánticos que resuelven el problema de Pattern Matching con una aceleración cuadrática respecto al estado del arte en la computación clásica. Es por esto que asegurando la existencia de un algoritmo cuántico que resuelve el problema de jumbled pattern matching, se hipotetiza que es posible diseñarlo de forma tal que su aceleración respecto al mejor algoritmo clásico es también cuadrática.

1.4. Objetivo general

Diseñar un algoritmo que resuelva el problema de jumbled matching bajo el paradigma de la computación cuántica obteniendo una ventaja en tiempo de ejecución respecto a los mejores algoritmos clásicos.

1.4.1. Objetivos especificos

- Encontrar los parámetros necesarios para la ejecución del algoritmo a diseñar.
- Construir los operadores y oráculos cuánticos con los cuales se diseñará el algoritmo.

Capítulo 2. Introducción a la computación cuántica

En la divulgación de la ciencia, la computación cuántica ha sido descrita como algo casi mágico, una herramienta capaz de resolver cualquier problema mejor y más rapidamente que las computadoras tradicionales; sin embargo, mucha información que circula a través de la red y que diseminan los medios de comunicación masivos sufre de ingenuidad, entendimiento equivocado y una buena dosis de sensacionalismo. Por esta razón, el primer capítulo de esta tesis intentará explicar la computación cuántica de la manera más conceptual y con el mayor cuidado posible.

2.1. Antecedentes históricos

2.1.1. Modelos de computación y tesis fuerte de Church-Turing

En 1937 Alan Turing publicó *On Computable Numbers, with an application to Entscheidungsproblem* (Turing, 1937) en donde proponía las *máquinas computables*, las cuales contaban con una cinta infinita dividida en celdas y un cabezal de lectura y escritura. Después de Turing, Kurt Gödel formuló un modelo basado en *funciones recursivas* y Alonzo Church hizo lo propio con el modelo llamado *cálculo lambda*. Prácticamente al mismo tiempo, Emil Post hizo avances independientemente de Turing con su modelo *procesos combinatorios finitos* en el cuál un trabajador se movía a lo largo de un espacio de trabajo infinito, con cajas que podrían ser marcadas o no marcadas, además dicho trabajador contaba con un conjunto de instrucciones .

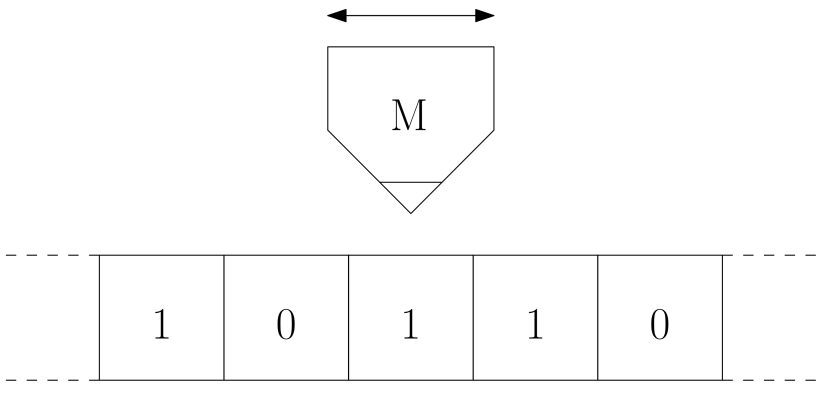

Figura 1. Una *máquina de Turing*, formada de un cabezal móvil que puede insertar y borrar caracteres sobre una banda infinita dividida por celdas.

Estos modelos resultaron ser equivalentes, es decir, cada uno es un tipo de máquina

de Turing, naciendo así la idea de una máquina universal, la cuál podría imitar el comportamiento de cualquier otra máquina de Turing. Finalmente, la prueba de que todos los modelos de computación clásica en competencia eran equivalentes, le permitió a Church proponer un principio, conocido como la tesis fuerte de Church-Turing:

Tesis fuerte de Church-Turing: *Cualquier proceso que es algorítmico en naturaleza define una función matemática que pertenece a una clase específica bien definida llamada funciones computables de Turing. En otras palabras, cualquier función computable por naturaleza puede ser calculada por una máquina de Turing*.

2.1.2. Definición formal de las Máquinas de Turing

Una MT opera un cabezal dispuesto sobre una cinta que tiene un inicio pero no tiene fin, extendiéndose a la derecha tanto como se requiera. Cada celda almacena un caracter y cuando el cabezal lee el caracter de la celda lleva a cabo una acción sobre la cinta (justo en la celda leída): cambiar el caracter por alguno nuevo, mover el cabezal a la izquierda o moverlo a la derecha. Existe además, un estado especial h, el cuál al ser leído por el cabezal hace que la computación de la MT se detenga; finalmente, si el cabezal llega más allá del inicio de la cinta (sin llegar a h) se dice que la máquina MT se *cuelga*.

Definición 1 *Una Máquina de Turing (MT) es una tupla* $M = (K, \Sigma, \delta, s)$ *donde*

- K *es un conjunto finito de estados,* $h \notin K$
- Σ *es un alfabeto finito,* $\# \in \Sigma$
- $s \in K$ *es el estado inicial*
- $\delta : K \times \Sigma \to (K \cup \{h\}) \times (\Sigma \cup \{\triangleleft, \triangleright\})$, $\triangleleft, \triangleright \notin \Sigma$ *es la función de transición*

En esta definición, la acción de moverse a la izquierda o derecha se denota por $\triangleleft$ y $\triangleright$ respectivamente. La acción de escribir # se llama también *borrar* la celda.
Con esta definición se puede describir el funcionamiento de las máquinas de Turing.

Ejemplo 1 *Una máquina de Turing que borra todas las letras α hacia la izquierda se escribe formalmente como $M = (K, \Sigma, \delta, s)$ con $K = \{0, 1\}, \Sigma = \{\alpha, \#\}, s = 0$, y la tabla a continuación describiendo la función de transición:*

δ	0	1
α	1#	$1, \alpha$
#	h, #	0, $\triangleleft$

Esta descripción compacta de la máquina de Turing se puede visualizar con un grafo donde los nodos son los estados y las transiciones están rotuladas. Una transición de p a q rotulada α, b significa que si la MT está en el estado p y hay una letra α bajo el cabezal, entonces pasa al estado q y realiza la acción b. El grafo que representa la MT que borra todas las α a la izquierda se puede ver en la Figura 2.

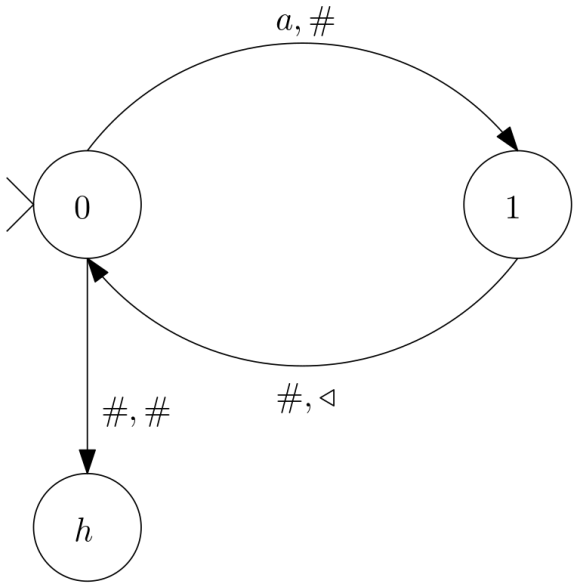

Figura 2. Grafo que representa la máquina de Turing que borra todas las α hacia la izquierda del cabezal, cuenta con dos estados, dos transiciones rotuladas y un estado de aceptación.

2.1.3. Máquina de Turing No Determinista

Las máquinas de Turing no deterministas son una extensión a las MT, donde dado un cierto estado y viendo un caracter bajo el cabezal, se tienen cero, una o más transiciones aplicables; además de poder elegir cualquiera de ellas. Si la máquina no tiene transiciones aplicables se cuelga, y en caso de tenerlas escoge alguna de ellas;

de forma que la máquina se detendrá con una cierta entrada si y sólo si existe una secuencia de elecciones que la lleven a la configuración detenida.

Definición 2 *Una Máquina de Turing no Determinista (MTND) es una tupla* $M = (K, \Sigma, \Delta, s)$*, donde* K, Σ *y* s *son como en la definición 1 y* $\Delta \subseteq (K \times \Sigma) \times ((K \bigcup \{h\}) \times (\Sigma \bigcup \{\triangleleft, \triangleright\}))$*.*

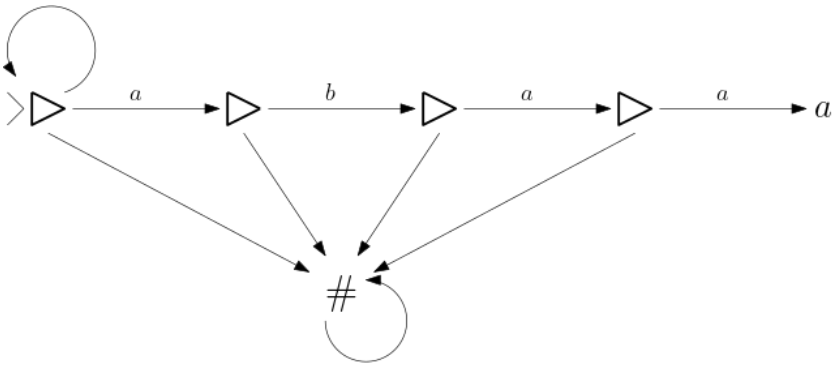

Figura 3. Máquina de Turing no Determinista que busca la secuencia *abaa* en la cinta.

A estas máquinas frecuentemente se les compara con un apostador perfecto, el cual siempre escoge las opciones que le harán ganar (la secuencia que le llevará al estado de aceptación).

2.1.4. Máquina de Turing probabilística

La descripción de las máquinas de Turing probabilísticas es parecida a la máquina de Turing no determinista, con la diferencia de que una vez que el cabezal lee un caracter, la elección de la transición no es *adivinada* o escogida *apropiadamente*, sino que cada una tiene cierta probabilidad de ser tomada; de modo que la probabilidad de seguir un cierto camino computacional es producto de la probabilidad de las transiciones usadas en dicho camino.

Definición 3 *Una Máquina de Turing Probabilística (MTP) tiene los mismos componentes que una MT. Es una tupla* $(K, \Sigma, q_0, q_f, \delta)$ *donde* K *es un conjunto finito de estados,* Σ *es un alfabeto finito,* $\# \in \Sigma$*,* $q_0 \in K$ *es el estado inicial,*$q_f \in K$ *es un estado final. La transición* δ *define una distribución de probabilidad según*

$$\delta : (K \times \Sigma) \times ((K \bigcup \{q_f\}) \times (\Sigma \bigcup \{\triangleleft, \triangleright\})) \rightarrow [0, 1].$$

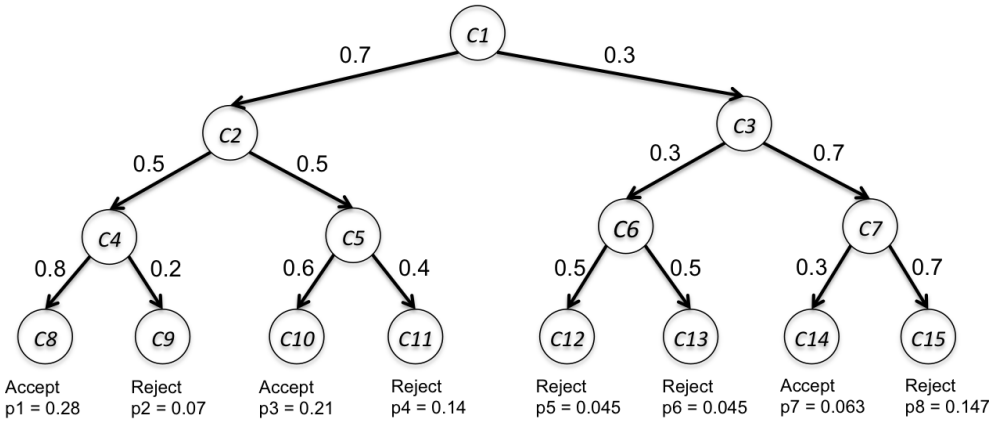

Figura 4. Árbol binario que representa una máquina de Turing probabilística, cada transición se toma con una cierta probabilidad, generando una distribución de probabilidad sobre los estados de aceptación y aquellos que no lo son. La probabilidad de seguir un cierto camino computacional es producto de la probabilidad de las transiciones usadas en dicho camino.

2.1.5. Máquina universal de Turing

Las máquinas de Turing se pueden pensar como casos específicos de otras más generales; se puede idear una máquina de Turing que sirva para sumar, otra que ordene elementos de una base de datos, y otra para hacer multiplicaciones. No obstante, si se cuenta con una máquina universal de Turing sería posible simular cualquiera de las anteriores, de este modo no sería necesario construir una máquina específica para cada programa a ejecutar; sino que esta máquina universal sería suficiente para llevar a cabo las tareas de todas las anteriores.

Definición 4 *Una* Máquina Universal de Turing *MTU recibe como entradas la descripción de una máquina de Turing MT y una entrada w, de modo que la MTU simulará el funcionamiento de la MT en forma tal que la simulación se detendrá, se colgará o entrará en ciclo infinito según MT lo haga con la entrada w. En caso de terminar, la cinta estará codificada en la misma forma en la que la MT codificaría su banda con la entrada w.*

2.1.6. Feynmann y la mecánica cuántica

La mecánica clásica aparenta ser causal, viendo al pasado se puede tomar en cuenta la posición y el momento de un sistema en dos tiempos distintos y predecir su futuro. Del mismo modo, si se conoce el estado actual de un sistema se puede saber desde

qué otro estado ha evolucionado. Es por tanto que la física clásica es causal, local y reversible. Feynmann se preguntaba en *Simulating physics with computers* (Feynman, 1982) si la física se podría simular en una computadora universal, específicamente si existe una forma exacta de simular la mecánica cuántica ya que las leyes de la naturaleza parecen ser bien representadas por este modelo. ¿Cómo se puede predecir con una computadora la naturaleza que parece ser probabilística? Es decir, dado un estado inicial como entrada de una computadora, calcular el estado final y que coincidiera con el estado final real de la naturaleza. ¿Es esto posible? Lamentablemente no, pues la naturaleza al ser probabilística es por tanto impredecible. Pero realmente no se necesita una simulación exacta de la naturaleza, es perfectamente aceptable tener un simulador probabilístico que posiblemente no pueda comportarse con exactitud en una sóla corrida como la naturaleza probabilística; sin embargo, si se deja correr el simulador una gran cantidad de veces, entonces este se acercaría con alta probabilidad al comportamiento (promedio) real de la naturaleza.

Entonces, ¿Cómo se debería abordar el problema de simular sistemas cuánticos? La primer forma es que la computadora en sí se construya con elementos mecánico cuánticos que obedezcan a las leyes de la física cuántica y por tanto ser posible realizar simulaciones de sistemas cuánticos. La segunda manera es usar las mismas computadoras clásicas de siempre, pero en este caso cabría preguntarse ¿Sobre estas computadoras se pueden realizar aquellas simulaciones cuánticas? y ¿Pueden ejecutarlas en tiempos razonables?

De estas preguntas primordiales fue que Feynman pensó por primera vez en la computadora cuántica como una forma plausible de computación universal. Ya que si estas computadoras cuánticas fueran capaces de simular los sistemas más fundamentales de la naturaleza, por consiguiente serían capaces de simular cualquier sistema clásico. Sucede que la segunda pregunta tiene respuesta negativa, esto pasa porque la dimensión del espacio de Hilbert donde viven los vectores de estado de un sistema cuántico (se tratará en secciones posteriores) se hace exponencialmente grande conforme se incrementan los grados de libertad del sistema; es decir, al aumentar en uno los grados de libertad se necesitaría el doble de parámetros para representar a di-

cho sistema cuántico en una computadora clásica. Por supuesto, la memoria necesaria resultaría ser ridículamente grande aún para sistemas con pocos grados de libertad. Esto a veces es tema de debate, ya que en casos muy particulares de sistemas con pérdida y ruido existen condiciones que permitirían realizar simulación clásica eficiente (Rahimi-Keshari *et al.*, 2016).

2.1.7. Tesis de Deutsch

El modelo de las máquinas de Turing es una buena abstracción matemática pero una pregunta válida sería saber si este modelo es consistente con la física conocida. Esta pregunta era irrelevante en la época de Turing debido a que las escalas de funcionamiento de las computadoras estaban muchos órdenes de magnitud por encima de los sistemas cuánticos; ahora los progresos en la miniaturización hace razonable reconsiderar los fundamentos de la computación (recordemos los transistores de escalas atómicas). Pese al esfuerzo de Turing por mantener la abstracción de las máquinas de Turing lejos de la física, algunos remanentes de la física clásica están intrínsecamente asociados a su modelo; especialmente la suposición de que un bit debe encontrarse en un estado definido (0 o 1); una idea probablemente concebida por simplicidad pensando en el determinismo que el universo aparenta tener a nuestra escala.

En 1985 David Deutsch propuso reformular la tesis de Church-Turing en términos físicos, siendo ahora consistente con las leyes físicas que mejor explicaban los experimentos de los últimos ochenta años: La mecánica cuántica (Deutsch, 1985).

Tesis de Deutsch: *Cualquier sistema físico finito realizable puede ser perfectamente simulado por un modelo universal de máquina de cómputo operado por medios finitos*.

La observación de Feynman de que un sistema cuántico se podría simular de forma eficiente (en tiempo polinomial) sólo mediante un modelo universal de cómputo basado en la mecánica cuántica inspiró a Deutsch, probando que era posible idear una *Máquina Universal de Turing Cuántica*.

2.1.8. Máquina de Turing cuántica

Definición 5 *Una máquina de Turing cuántica (MTQ) se define por una tripleta* (Σ, Q, δ)*, donde* Σ *es un alfabeto finito,* $\# \in \Sigma$*,* Q *es un conjunto finito de estados con un estado inicial* q_0 *y un estado final* $q_f \neq q_0$*, y* δ*, la función de transición cuántica*

$$\delta : Q \times \Sigma \to \tilde{\mathbb{C}}^{\Sigma \times Q \times \{L,R\}} \tag{1}$$

donde $\tilde{\mathbb{C}}$ *es el conjunto de* $\alpha \in \mathbb{C}$ *tal que existe un algoritmo determinista que calcula las partes real e imaginaria de* α *hasta* 2^{-n} *en tiempo polinomial en* n*.*

La MTQ tiene una cinta infinita de celdas indexadas por $\mathbb{Z}$ *y un cabezal que se mueve a lo largo de la cinta. Las configuraciones intermedias, iniciales y finales se definen exactamente como para las TM determinísticas. Sea* S *el espacio con producto interno de combinaciones lineales complejas finitas de configuraciones de* M *con la norma Euclidiana. Se denota como una superposición de* M *a cada elemento* $\phi \in S$ *. La MTQ* M *define un operador lineal* $U_M : S \to S$*, llamado el* operador de evolución temporal de M*, como sigue: si* M *comienza en una configuración* c *con estado* p *y símbolo escaneado* σ*, entonces después de un paso* M *estará en superposición de configuraciones* $\Psi = \Sigma_i \alpha_i c_i$*, donde cada* α_i *no cero corresponde a una transición* $\delta(p, \sigma, \tau, q, d)$*, y* c_i *es la nueva configuración que resulta de aplicar esta transición a* c*. Al extender este mapeo a todo el espacio* S *a través de la linearidad, obtenemos el operador de evolución temporal lineal* U_M*.*

Notemos que S está definido sobre una base ortonormal: las configuraciones de M. En términos de esta base estándar, cada superposición $\psi \in S$ se podría representar como un vector de números complejos indexados por configuraciones. El operador de evolución temporal U_M se podría representar por la matriz cuadrada (dimensionalmente contable) con filas y columnas indexadas por configuraciones donde el elemento matricial de la columna c y fila c' define la amplitud con la que la configuración c conduce a la configuración c' en un solo paso de M.

2.1.9. Diferencia entre las máquinas de Turing cuánticas (MTQ) y máquinas de Turing probabilísticas (MTP)

Hay que aclarar que las MTP y las MTQ no son equivalentes. Se puede pensar mejor a las MTQ como una generalización de las MTP. En una MTP, si se inicializa la cinta en alguna configuración inicial y se corre la máquina sin inspeccionar su estado en t pasos, entonces su estado final será incierto y se puede describir sólo con una distribución de probabilidad sobre todos los estados accesibles en t pasos. Por otro lado, en una QTM si se inicia la máquina en alguna configuración inicial y se le permite evolucionar t pasos, entonces su estado lo describe una superposición de todos los estados alcanzables en t pasos. La diferencia radica en que una MTP clásica sólo sigue una trayectoria computacional particular; es decir, aunque el estado final depende de una distribución de probabilidad, sólo existe un único camino que la máquina ha seguido a través de los t pasos, pero que es desconocido. En el caso de las MTQ, después de los t pasos, cada una de las posibles trayectorias computacionales existirá dentro de una superposición (Deutsch, 1985), la cuál generará una trayectoria bien definida hasta que se realice la inspección de la máquina. En resumen, el desconocimiento de la trayectoria computacional no implica que esta no esté definida en una máquina de Turing probabilística, mientras que en la máquina de Turing cuántica el propio desconocimiento de la configuración de la máquina implica que no existe una trayectoria computacional definida, y cuando se inspecciona o se *mide* a la máquina, esta acción *borra* todas las trayectorias computacionales excepto una, justo aquella que resultará de dicha medición.

El proceso de medición se puede entender como una forma de forzar a la máquina a *escoger* alguna de las trayectorias que se encuentra en superposición o combinación lineal, donde cada una de esas trayectorias tendrá asociado un parámetro que definirá la probabilidad de ser *escogida*. Si se corriera m veces una máquina de Turing cuántica Q, cada vez con la misma función de transición, se mediría una trayectoria l_j en $m * \frac{1}{P_{l_j}}$ veces, donde P_{l_j} es la probabilidad de medición asociada a l_j. Como se verá más adelante es posible modificar esos parámetros llamados *amplitudes de probabilidad* para hacer más o menos probable la medición de ciertas trayectorias.

El modelo de las máquinas de Turing cuánticas se puede utilizar para realizar análisis de complejidad y construcción de algoritmos; sin embargo, es un modelo de cómputo físicamente no realizable, ya que los cabezales no se pueden encontrar en superposición por ser un sistema físico macroscópico. Por tanto, para esta tesis se utiliza el modelo de los *circuitos cuánticos*, ya que en 1993 Yao mostró que cualquier función computable por una máquina de Turing cuántica es equivalente a un circuito cuántico de tamaño polinomial (Yao, 1993). En secciones posteriores se verá la construcción de estos circuitos.

2.2. El modelo de la computación cuántica

Antes de estudiar la descripción de los algoritmos fundamentales para este trabajo, hay que repasar los conceptos teóricos fundamentales de la computación cuántica. Especialmente los relacionados con la representación y manipulación de datos, se discuten estos tópicos con la mayor claridad posible sin profundizar en definiciones y demostraciones formales.

2.2.1. Bits y Qubits

El componente principal de la computación cuántica es el qubit, que es el análogo al bit clásico. La construcción de un qubit depende de un sistema cuántico de dos estados; sin embargo, para efectos de la teoría algorítmica el tratar a los qubits como objetos matemáticos alejados de la física da una libertad para construir una teoría general de la computación e información cuántica que no depende de la arquitectura o realización física de un sistema (Nielsen y Chuang, 2010). Tal como un bit, los qubits tienen dos posibles estados que usando la notación convencional de Dirac se escriben como los estados base $|0\rangle$ y $|1\rangle$. Mientras los bits tienen estados definidos 0 o 1, los qubits pueden tener estados definidos y comportarse como bits o encontrarse en estados no definidos llamados *estados en superposición* o simplemente *superposiciones*.

2.2.2. Esfera de Bloch y espacios de Hilbert

Una representación geométrica muy útil de un qubit de dos estados es la *esfera de Bloch*, dicha esfera tiene un radio unitario y vive en un espacio tridimensional, esta representación gráfica da una idea conceptual del comportamiento tan distinto que tiene el qubit en comparación con su análogo clásico. El estado cuántico de un qubit se representa como un vector en la superficie de la esfera, cuando este vector apunta hacia los polos norte o sur de la esfera el vector se encontrará en los estados puros $|0\rangle$ y $|1\rangle$ respectivamente, en otras palabras, estos polos representan los eigen-estados $|0\rangle$ y $|1\rangle$.

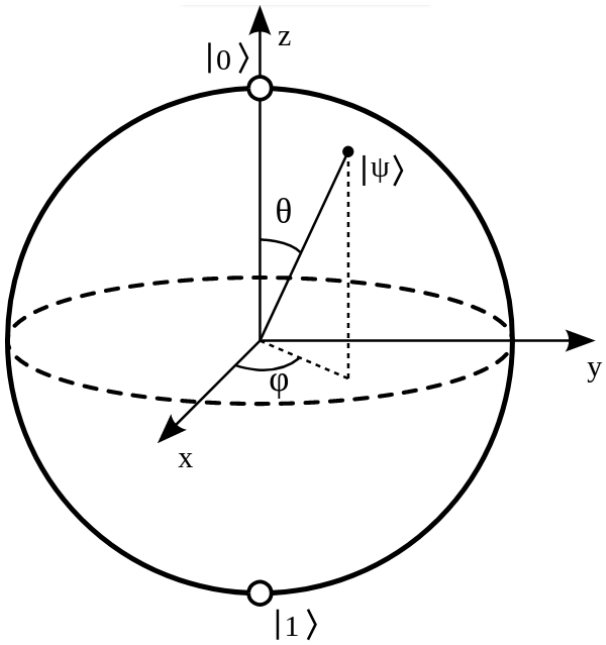

Figura 5. Representación gráfica de un qubit de dos estados.

Un estado cuántico general de un qubit se puede escribir en términos de los ángulos azimutal y de elevación, como se muestra en la Ecuación 2:

$$|\psi\rangle = \cos(\theta/2)|0\rangle + e^{i\phi}\sin(\theta/2)|1\rangle \tag{2}$$

donde $0 \leq \theta \leq \pi$ y $0 \leq \phi \leq 2\pi$. A este estado general también se le llama *superposición*, y a los sistemas que tienen dos o más qubits se les llama *registros multi-qubits o registros de memoria cuántica* y una posible representación gráfica de estos registros

es la misma esfera de Bloch pero repetida en tantos qubits existan en el sistema.

En resumen, un qubit se encuentra en general en un estado de la forma mostrada en la Expresión 3:

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle \tag{3}$$

donde los estados (kets o vectores) $|0\rangle$ y $|1\rangle$ son la base de un espacio de Hilbert que es un espacio complejo dotado de producto interno. Aunque en la mecánica cuántica los vectores que viven en este espacio de Hilbert son representaciones abstractas del estado de un sistema físico, en este trabajo es suficiente decir que estos vectores son los posibles estados en los que se encuentra al registro de memoria. Si α y β son ambos distintos de 0, entonces el qubit se encuentra en un estado de superposición, en caso contrario el qubit tendrá un estado definido. A esta base se le llama la *base computacional* y si se usan registros con más de un qubit se formarán distintos espacios compuestos, por ejemplo espacios donde viven vectores como los de la Expresión 4.

$$|0110\rangle, |11111\rangle, |10\rangle \tag{4}$$

Para construir estos estados base o eigenvectores de espacios de Hilbert, se hace uso de la operación ”⊗” llamada producto tensorial; que aplicada entre espacios de Hilbert básicamente es un producto cartesiano, y aplicada sobre sobre vectores es un producto externo.

Ejemplo: Sean los vectores $|0\rangle$ y $|1\rangle$ de la base computacional

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

y dos espacios de Hilbert $\mathbb{H}_1$ y $\mathbb{H}_2$ idénticos, cuyos vectores base o eigen vectores son $|0\rangle$ y $|1\rangle$, se puede formar un nuevo espacio de Hilbert $\mathbb{H}_3 = \mathbb{H}_1 \otimes \mathbb{H}_2$ cuyos elementos son los productos tensoriales de los eigen vectores de cada uno de los espacios; es decir, elementos como los mostrados en la Ecuación 5.

$$|0\rangle \otimes |0\rangle, |0\rangle \otimes |1\rangle, |1\rangle \otimes |0\rangle \text{ y } |1\rangle \otimes |1\rangle$$

$$= \begin{pmatrix}1\\0\end{pmatrix} \otimes \begin{pmatrix}1\\0\end{pmatrix}, \begin{pmatrix}1\\0\end{pmatrix} \otimes \begin{pmatrix}0\\1\end{pmatrix}, \begin{pmatrix}0\\1\end{pmatrix} \otimes \begin{pmatrix}1\\0\end{pmatrix}, \begin{pmatrix}0\\1\end{pmatrix} \otimes \begin{pmatrix}0\\1\end{pmatrix} \tag{5}$$

$$\begin{pmatrix}1\\0\\0\\0\end{pmatrix}, \begin{pmatrix}0\\1\\0\\0\end{pmatrix}, \begin{pmatrix}0\\0\\1\\0\end{pmatrix}, \begin{pmatrix}0\\0\\0\\1\end{pmatrix}$$

Los eigen estados de este espacio de Hilbert se escriben como cadenas dentro de los kets o vectores, algunos ejemplos se muestran en la Expresión 6

$$|1000\rangle, |0100\rangle, |0010\rangle, |0001\rangle. \tag{6}$$

La dimensión del nuevo espacio de Hilbert es 4, conforme se usan más qubits para formar estos espacios de Hilbert, se aumentarán los estados base o eigen vectores en una cantidad exponencial.

2.2.3. Superposición

Anteriormente se señaló que los qubits tienen dos estados posibles $|0\rangle$ y $|1\rangle$ y que además pueden estar en estados no definidos llamados *superposiciones*. En el espacio de Hilbert existen combinaciones lineales de los estados puros o eigen-estados, y justo estas combinaciones lineales son las superposiciones escritas como en la Ecuación 7

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle, \tag{7}$$

donde los números α y β son números complejos y cumplen la propiedad de la Ecuación 8

$$|\alpha|^2 + |\beta|^2 = 1, \tag{8}$$

porque, como se discuté más adelante, estos números son parámetros de una distribución de probabilidad. Así, podríamos hacer una generalización y tener espacios vectoriales de $N = 2^n$ eigen-estados

$$|\psi\rangle = \sum_{i=0}^{2^n-1} c_i |i\rangle, \tag{9}$$

donde se debe cumplir la Ecuación 10, que es la propiedad generalizada de la Ecuación 8 para los N estados.

$$\sum_{i=0}^{2^n-1} |c_i|^2 = 1 \tag{10}$$

El sistema de qubits descrito por la Ecuación 9 se encuentra en una superposición de estados que es una generalización de la Ecuación 7. Esta superposición es la que produce un atributo de la computación cuántica que no está presente en la computación clásica: el *paralelismo cuántico*, que se explicará con detalle más adelante.

2.2.4. El fenómeno del entrelazamiento

A veces se entiende al entrelazamiento cuántico como un efecto por el cual dos partículas se pueden comunicar instantáneamente o que al modificar una de ellas la otra se modifica de igual forma e inmediatamente; pero habrá que aclarar un par de cosas sobre este efecto que hasta al mismo Einstein le causó conflictos y que lamentablemente aún no se ha entendido completamente entre los físicos teóricos.

En la mecánica cuántica se dice que dos partículas se encuentran *entrelazadas* cuando sus estados cuánticos no se pueden describir independientemente; es decir, dos partículas en entrelazamiento tienen una función de onda única que las describe al mismo tiempo.

Supóngase una partícula A que se decribe por $|\psi\rangle_A$ y otra partícula B descrita por $|\psi\rangle_B$, al estado unión correspondiente al sistema creado por estas partículas se le denota como $|\psi\rangle_{A,B}$. Si estas partículas se encontraran en un estado de entrelazamiento, sería imposible escribir dicho estado como el producto directo de sus estados cuánticos independientes, como en la Desigualdad 11.

$$|\psi\rangle_{A,B} \neq |\psi\rangle_A \otimes |\psi\rangle_B \tag{11}$$

Esto significa que a estas dos partículas no se les puede describir de forma independiente y además que existe una correlación entre ambas. Al realizar ciertas acciones sobre una de las partículas, por esta correlación la otra inmediatamente cambiará su

estado, aún cuando aquellas acciones no se hallan aplicado directamente a la última partícula. Einstein le llamó a este fenómeno *acción fantasmal a distancia* ya que no se necesitaba ningún medio para que una partícula se comunicara con otra aún cuando las pudiera separar una distancia inmensa (digamos, de millones de años luz).

Un punto importante por aclarar es que esta acción a distancia aunque es inmediata y que pareciera superar a la velocidad de la luz como un medio de transmisión de información en realidad no se puede usar para tal fin, o al menos no sin un canal o medio clásico. Esto pasa ya que aún cuando la acción es inmediata, si se supone que dos interlocutores se encuentran cada uno junto a su respectiva partícula entrelazada; cuando uno de los dos midiera (perturbar físicamente para terminar con el entrelazamiento) su partícula, el otro no sabría de ninguna forma que su partícula ya habría cambiado al estado que le corresponde debido a la correlación que tiene con la primera partícula. Para que el segundo interlocutor supiera que ya puede observar su partícula necesitaría un aviso del primer interlocutor, pero dicho aviso no puede viajar más rápido que la velocidad de la luz. Y si el segundo interlocutor quisiera observar su partícula antes de tiempo no sabría si el estado en el que la encontrase es causado por el entrelazamiento o por la misma acción de observar a la partícula. Por tanto, es imposible obtener información útil del entrelazamiento con una velocidad mayor a la de la luz.

En la computación cuántica el entrelazamiento se da entre subconjuntos de qubits, y cuando estos se encuentran entrelazados al aplicar acciones sobre un subconjunto se puede impactar directamente sobre otros aún sin manipularlos físicamente. Por ejemplo, si se tiene un sistema de dos qubits A y B entrelazados en el siguiente estado de la Ecuación 12

$$|\psi\rangle_{A,B} = \frac{1}{\sqrt{2}}(|0\rangle_A \otimes |1\rangle_B + |1\rangle_A \otimes |0\rangle_B) \tag{12}$$

Si al medir u observar el estado del qubit A se encuentra, por ejemplo en $|0\rangle$, automáticamente el qubit B tomaría el estado $|1\rangle$. Caso contrario, si el estado del qubit A fuera $|1\rangle$, automáticamente el qubit B tomaría el estado $|0\rangle$.

2.2.5. Mediciones

Una vez descritos los dos principales componentes que diferencian a la computación clásica de la cuántica, se debe saber qué sucede cuando se mide un registro de memoria cuántico.

Un sistema cuántico en la vida real es muy delicado; prácticamente cualquier interacción con el entorno puede causar pérdida de los estados en superposición y por tanto de entrelazamiento. Como ejemplo consideremos un sistema de qubits hechos de superconductores, si no se logra mantener en el rango correcto la temperatura, podría suceder que los qubits en entrelazamiento decayeran a un estado físico bien definido (alguno de los eigen-estados), perdiendo su estado de entrelazamiento. A este fenómeno se le llama *decoherencia cuántica*. Para efectos del presente trabajo no se tomarán en cuenta esas posiblidades. Hacer una medición es básicamente realizar una *decoherencia cuántica controlada*; hablando desde el punto de vista de la teoría de la computación cuántica y muy conceptualmente, sería como tomar un cierto conjunto de los qubits que forman el registro de memoria cuántico y hacer que decayeran a uno de los eigen-estados con algún tipo de perturbación pertinente (cómo se hace esto no es algo que importe en este trabajo).

En resumen, una vez procesado el registro, hacer una *medición* de éste es aplicar una decoherencia cuántica controlada. El eigen-estado al que decaiga está dado por la función de onda o distribución de probabilidad del sistema.

2.2.6. Evolución de los registros de memoria

Para que los registros de memoria sean útiles como medio de cómputo se requiere una forma de hacerlos evolucionar. Como el registro de memoria es un sistema cuántico aislado, su comportamiento se debe describir por la Ecuación 13, llamada la *ecuación de Schrödinger*:

$$i\hbar \frac{\partial |\psi(t)\rangle}{\partial t} = H|\psi(t)\rangle\,, \tag{13}$$

donde H es llamado el Hamiltoniano y contiene todos los detalles del sistema físico.

Cuando se conoce el Hamiltoniano se puede dar solución a la ecuación de Schrödinger, cuya solución más simple para un Hamiltoniano independiente del tiempo está dada por la Ecuación 14.

$$U(t) = \exp(-iHt/\hbar) \tag{14}$$

Con lo que si se conoce el estado inicial del sistema se puede determinar el estado al tiempo t al hacer actuar el operador $\exp(-iHt/\hbar)$ sobre el estado inicial. En otras palabras, el sistema descrito por algún Hamiltoniano H, cuando se deja *correr* un tiempo t, el resultado que se obtiene está dado por la Ecuación 15:

$$|\psi(t)\rangle = U(t)\,|\psi(0)\rangle = \exp(-iHt/\hbar)\,|\psi(0)\rangle\,, \tag{15}$$

como H es una matriz hermitiana, su matriz exponencial es una matriz *unitaria* con la propiedad de que su inversa es igual a su conjugada $U^{-1} = U^{\dagger}$, permitiendo una evolución reversible, por tanto, sin pérdida de información. Se puede decir que el análogo clásico más cercano a la computación cuántica es la computación clásica reversible ya que también preserva información acerca de la historia computacional.

Resumiendo, la manipulación de los qubits se realiza con los operadores U, estos actúan sobre el registro para evolucionarlo, dichos operadores de evolución se representan por matrices unitarias. Por ejemplo, el análogo cuántico de la compuerta NOT se escribe como en la matriz de la Ecuación 16:

$$\sqrt{NOT} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & -1 \\ 1 & 1 \end{pmatrix} \tag{16}$$

Al aplicar dos veces esta compuerta a un qubit, realizará una inversión. La primera aplicación de la Ecuación 17 cambia el estado, pero la aplicación de la Ecuación 18 lo devuelve al estado original con la fase invertida

$$\sqrt{NOT}[\,\sqrt{NOT}\,|0\rangle\,] \implies \sqrt{NOT}\left[\left[\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}\right]^{T}\right] \implies |1\rangle \tag{17}$$

y

$$\sqrt{NOT}[\sqrt{NOT}\,|1\rangle] \implies \sqrt{NOT}\left[\left[-\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}\right]^T\right] \implies -1\,|0\rangle \tag{18}$$

2.2.7. Interpretación computacional

Una abstracción de un ciclo que tendría lugar en una computadora cuántica puede resumirse en un proceso *PREPARAR-EVOLUCIONAR-MEDIR* que es el análogo al proceso abstracto que sigue una computadora clásica *CARGAR-CORRER-LEER* donde uno carga los datos a la máquina, corre un progama que usa los datos como entrada, y finalmente lee el resultado.

2.3. Modelo de los circuitos cuánticos

La descripción de algoritmos cuánticos se realiza con un *modelo circuital* que provee una representación visual de cómo una compleja computación cuántica de multi-qubits se descompone en una secuencia de compuertas más simples. Los circuitos son líneas horizontales donde el qubit más significativo está en la línea superior y el menos significativo en la línea del fondo. El tiempo fluye desde la izquierda hacia la derecha en el circuito y una vez procesado el registro se mide el circuito con una *compuerta de medición*. En este último paso se mide el qubit para obtener un valor clásico.

Teniendo inicializado nuestro registro en el estado

$$|\psi\rangle = \sum_{i=0}^{2^n-1} c_i\,|i\rangle\,,$$

un *algoritmo cuántico* es, en pocas palabras, una sucesión de operadores que *evolucionarán* el registro apropiadamente, justo como en la Ecuación 15, al resultado de la aplicación de un operador se le denota como en la Ecuación 19:

$$|\psi_{k+1}\rangle = U_k\,|\psi_k\rangle\,, \tag{19}$$

donde U_k es una compuerta unitaria y $|\psi_{k+1}\rangle$ es el estado después de aplicar la compuerta. Este proceso se puede ejemplificar con un algoritmo muy simple que se describe a continuación.

Considérese un registro de memoria de dos qubits q_1 y q_2, que en conjunto se les llama $|\psi\rangle$, tal vez construidos con trampas de iones, superconductores o algún sistema óptico. Para una mejor visualización se usan puntos que simulen ser qubits.

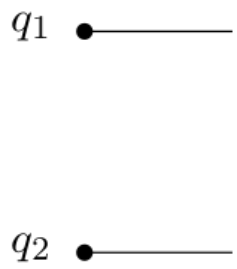

Figura 6. Visualización de un registro de memoria de dos qubits como un circuito cuántico.

Luego, se quiere utilizar estos qubits para hacer una tarea computacional, entonces se genera un estado en superposición que es el arma secreta de la computación cuántica. Aquello que generará esta superposición se le llama *compuerta generadora de superposiciones* y en el esquema se pondrá justo después del registro de memoria. En el ejemplo de la Figura 7 a esta compuerta se le denota por H, llamada compuerta de Walsh-Hadamard; aunque existen más compuertas generadoras de superposiciones, la de Hadamard generalmente se usa para la descripción de una gran variedad de algoritmos cuánticos. La compuerta de Hadamard actúa sobre un qubit asignando estados de superposición a los eigen-estados $|0\rangle$ y $|1\rangle$ como

$$H|0\rangle = \frac{|0\rangle+|1\rangle}{\sqrt{2}}$$
$$H|1\rangle = \frac{|0\rangle-|1\rangle}{\sqrt{2}}.$$

Podemos generalizar este operador como $H^{\otimes N}$ para que actue sobre un registro de memoria de N qubits, así para cada uno de los 2^N estados posibles de este espacio el operador generará una superposición distinta.

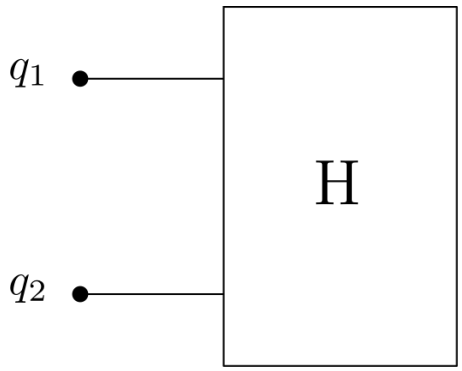

Figura 7. Las compuertas reciben como entrada el registro de memoria, esto se puede visualizar como dos líneas que entran en *H*.

Ahora la superposición se usará en una tarea computacional, a esta tarea se le llama *operador cuántico*, que en este caso es una caja negra. Esta caja recibe como entrada el registro y como salida se obtendrá el mismo registro pero modificado por el operador. Estos operadores son conjuntos de compuertas que hacen evolucionar el registro, veremos ejemplos de estos en la subsección dedicada al *algoritmo de Grover*. Esta compuerta se agrega a nuestro circuito de la Figura 8 y se le denota por *T*.

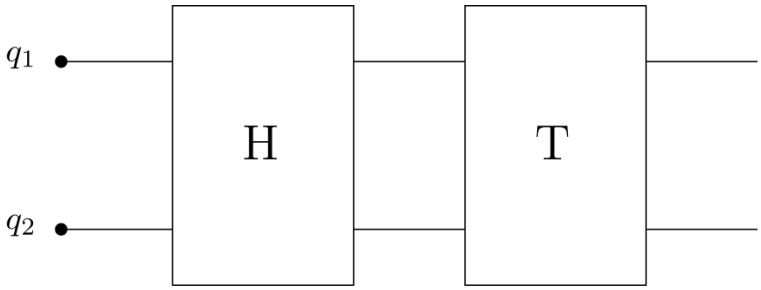

Figura 8. Después de pasar a través de *H* el registro de memoria se encontrará en un estado de superposición y pasará a través de la compuerta *T*, en esta compuerta se realizará un cálculo dado y después el registro modificado será la salida de *T*.

Finalmente al decidir *medir el sistema* o como se mencionó anteriormente, se aplicará una decoherencia cuántica controlada para que la *función de onda* que representa este sistema decaiga a un estado base. Con lo cuál ahora el registro de memoria será información clásica bien definida. El circuito que representa el algoritmo descrito se puede observar en la Figura 9.

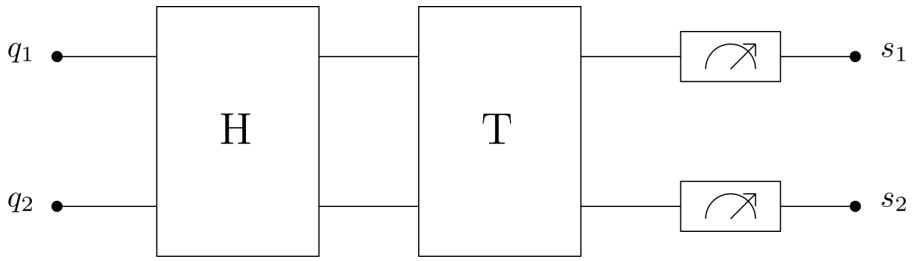

Figura 9. Circuito que representa un algoritmo cuántico completo con una compuerta *H* generadora de superposiciones, un operador *T*, y mediciones indicadas por las cajas con un símbolo parecido a un tacómetro.

2.4. Paralelismo cuántico

Al discutir la diferencia entre máquinas de Turing probabilísticas y cuánticas, se mencionó que las primeras toman un camino computacional definido pero cada paso es probabilístico, mientras que la segunda toma todos los caminos computacionales al mismo tiempo y hasta no inspeccionar el estado de la máquina, el camino computacional está indefinido. Una forma de visualizar esta diferencia es con un par de árboles de decisión.

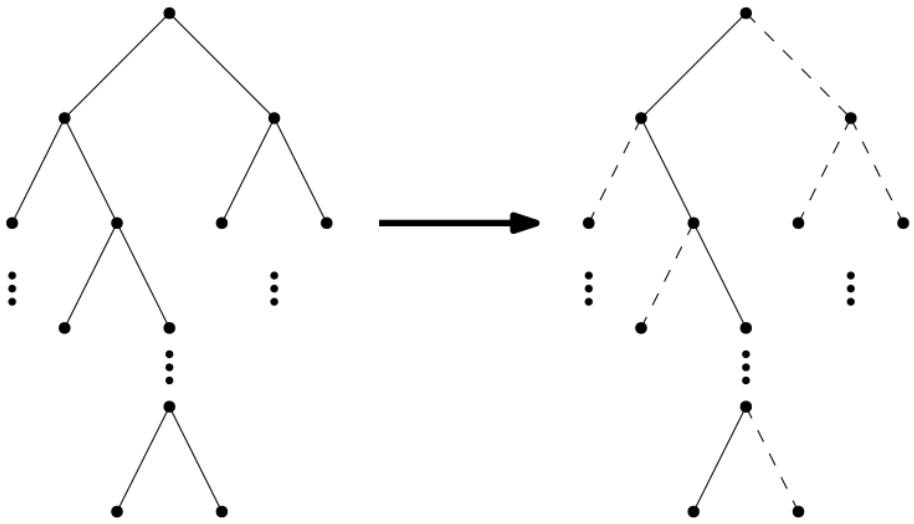

Figura 10. La máquina de Turing probabilística toma uno de los posibles caminos computacionales aún cuando no se haya examinado la configuración de la máquina, la configuración de la máquina está dada por una distribución de probabilidad

El árbol binario de la Figura 10 es una esquematización de lo que sucede con las máquinas de Turing probabilísticas, aunque la configuración sin inspeccionar de la máquina después de *t* pasos está dada por una distribución de probabilidad, en realidad

la máquina ha tomado un sólo camino computacional sin importar que sea desconocido; es decir, las ramificaciones que llevarían a una de las posibles configuraciones en t pasos en realidad nunca se tomaron, y sólo existen en la distribución de probabilidad como abstracciones. Por otro lado, los árboles correspondientes a las máquinas de turing cuánticas mantienen las ramificaciones en superposición, permitiendo aplicar *operadores* en cada uno de los caminos computacionales al mismo tiempo. En un sólo paso podemos realizar 2^n operaciones, donde n es la altura del árbol; a esta característica única de la computación cuántica se le llama *paralelismo cuántico* (Ver Figura 11).

Finalmente, cuando se inspecciona la máquina, la *función de onda* que representa al registro de memoria sufre una *decoherencia* decayendo hacia alguno de los caminos computacionales con probabilidad $|c_j|^2$, eliminando los otros caminos. No hay que confundirse, en las máquinas de Turing probabilísticas nunca se tomó ninguno de los otros caminos, sólo se ignoraba el avance de la máquina al no inspeccionarla; por otro lado, en la máquina cuántica realmente se *tomaron* todas las ramificaciones al mismo tiempo y el camino final surgió al momento de inspeccionar la máquina.

2.5. Búsqueda no estructurada y algoritmo de Grover

Después de haber revisado los fundamentos de la computación cuántica se está en condiciones de presentar al algoritmo de Grover, uno de los dos algoritmos pioneros en la computación cuántica. El algoritmo de Grover resuelve el problema de la búsqueda no estructurada, donde se tiene una base de datos desordenada y se desea encontrar un elemento particular. Una computadora clásica no puede hacer nada mejor que verificar uno a uno los registros de la base de datos hasta encontrar el elemento buscado, con lo que en tiempo promedio tardará $N/2$ pasos en resolver el problema, o en peor tiempo $O(N)$, donde N es el tamaño de la base de datos. Por otro lado, una computadora cuántica mediante la superposición y el entrelazamiento tardaría $O(\sqrt{N})$ pasos para encontrar el elemento deseado, es decir, con una aceleración cuadrática respecto a la solución clásica.

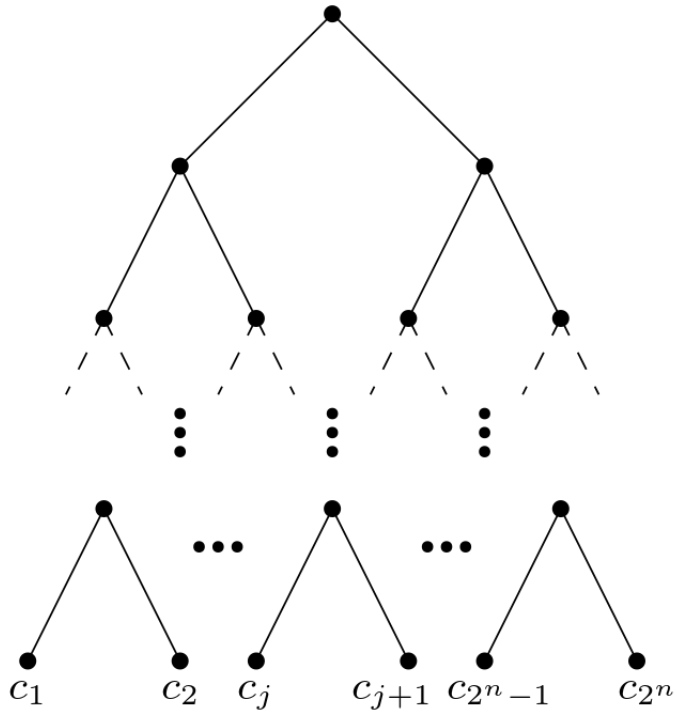

Figura 11. La máquina de Turing cuántica mantiene en superposición todos los caminos computacionales, y al inspeccionar su configuración se mide alguno de los caminos con probabilidad $|c_j|^2$.

Otro algoritmo fundamental en la computación cuántica es el de Shor, diseñado para factorizar números enteros compuestos (enteros que son el producto de dos números primos); mientras que en una computadora clásica los algoritmos del estado del arte utilizan tiempo superpolinomial, siendo imprácticos cuando tales números rebasan los 100 dígitos. El algoritmo de Shor resuelve el problema en un tiempo que crece polilogarítmicamente en el tamaño del número a ser factorizado. Como para este trabajo no es de utilidad el algoritmo de Shor, no se profundiza en su descripción y análisis.

2.5.1. El problema de la búsqueda no estructurada

El problema de búsqueda no estructurada se define en una forma que sea posible estimar el costo computacional de resolver el problema clásica y cuánticamente, con el fin de facilitar su comparación.

Definición 6 *Considérese un problema de búsqueda donde se requiere encontrar un elemento particular de entre un conjunto de N candidatos. Se supone que esos N candidatos están etiquetados por índices x en el rango* $0 \leq x \leq N-1$*, y el índice del*

elemento buscado es $x = t$*. Sea un oráculo computacional, de una* función black-box*,* $f_t(x)$ *que cuando recibe como entrada el elemento de índice* x *puede decidir si dicho índice es el objetivo buscado. Es decir, el oráculo se define mediante la Ecuación 20:*

$$f_t(x) = \begin{cases} 0 \quad si \quad x \neq t \\ \\ 1 \quad si \quad x = t \end{cases} \tag{20}$$

donde 0 *es* no *y* 1 *es* si*. El problema es* no estructurado *pues no hay un patrón discernible para los valores de* $f_t(x)$ *que den alguna guía para encontrar* $x = t$*. Nuestro trabajo es encontrar el índice* $x = t$ *usando la menor cantidad de llamadas al oráculo* $f_t(x)$*.*

Es posible definir estas funciones sin dar la *receta* de cómo construirlas en concreto. Ese problema está relacionado a la implementabilidad del oráculo y no a la construcción de los algoritmos en sí, veremos más a detalle aspectos relacionados a los oráculos en una sección posterior sobre complejidad cuántica.

2.5.2. Solución Clásica

Muchos algoritmos clásicos aceptan soluciones de tipo generación y prueba; es decir, en el espacio de búsqueda se genera un caso específico y se prueba si éste satisface las restricciones. Resulta natural utilizar la notación de Dirac para describir estos algoritmos que incidentalmente coincide con la notación que se utiliza para describir algoritmos cuánticos. Supóngase que se tiene una bolsa de N índices y que se busca repetidamente dentro de la bolsa, al tomar un índice se le pregunta al oráculo si es el objetivo, $|t\rangle$. Si es el elemento, se para el proceso. Si no, se regresa el índice de vuelta en la bolsa y se repite el proceso. En cada repetición la probabilidad de encontrar el índice buscado es de $1/N$, de modo que se necesita realizar este proceso aproximadamente N veces para encontrar la solución con probabilidad $O(1)$. Se puede imitar este procedimiento en una computadora cuántica. Una bolsa de índices cuántica puede ser considerada como una superposición equlibrada (donde cada estado base tiene la misma amplitud) de todos los índices en el rango $0 \leq x \leq N-1$, es decir,

un estado como el de la Ecuación 21.

$$|\psi\rangle = \frac{1}{\sqrt{N}} \Sigma_{x=0}^{N-1} |x\rangle \tag{21}$$

Similarmente, el análogo cuántico de la toma aleatoria de un índice dentro de la bolsa es la medición de esta superposición en la base de los índices. Al medir, se obtendrá algún $|x\rangle$ particular; entonces se le pregunta al oráculo si es el índice $x = t$. Como las amplitudes son iguales en cada estado base, al realizar la medición la probabilidad de encontrar nuestro objetivo es $1/N$; por tanto, es necesario repetir este procedimiento N veces para encontrar la solución con probabilidad $O(1)$, exactamente como en la solución clásica.

2.5.3. Algoritmo de Grover

Una computadora cuántica no está limitada a probar cada índice en sucesión. En vez de eso, puede probar todos los índices de una sóla vez, en superposición, usando el paralelismo cuántico. Lamentablemente no se puede ver esos resultados para cada prueba individualmente debido a la decoherencia cuántica (recordemos que la decoherencia cuántica es la pérdida del entrelazamiento), el paralelismo cuántico no confiere ninguna ventaja por sí mismo.

En 1996 Lov Grover descubrió la técnica de la *amplificación de amplitud*, donde la función oráculo definida por la Ecuación 20 es utilizada para crear un operador que incrementa la amplitud del índice objetivo y decrementa la amplitud de aquellos que no lo son. De este modo, al crear una superposición de todos los posibles índices y después amplificar la amplitud de los elementos objetivo, se puede sesgar la salida de la medición hacia estos elementos, que ahora tendrán una mayor probabilidad de ser medidos.

El operador de amplificación tiene la forma

$$Q = -U\mathbb{1}_s U^\dagger \mathbb{1}_t,$$

donde $\mathbb{1}_s = \mathbb{1} - 2|s\rangle\langle s|$, es el operador que invierte la fase del estado inicial $|s\rangle$, $\mathbb{1}_t = \mathbb{1} - 2|t\rangle\langle t|$ es el operador que invierte la fase del estado objetivo $|t\rangle$, y el operador U es el que genera una superposición en la que se garantice que contiene un componente no nulo en el estado objetivo t. Este operador generalmente es el operador Walsh-Hadamard, que genera una superposición equilibrada.

El algoritmo de Grover se puede resumir en los siguientes pasos:

1. Dado un oráculo, o función de caja negra $f_t(x)$ que puede decidir si un índice dado x es el índice buscado t, se construye el operador de *amplificación de amplitud* $Q = -U\mathbb{1}_s U^\dagger \mathbb{1}_t$ usando la función black-box $f_t(x)$ donde:
   - $|s\rangle$ = el estado inicial del registro
   - $|t\rangle$ = el estado objetivo (desconocido)
   - $\mathbb{1}_s = \mathbb{1} - 2|s\rangle\langle s|$
   - $\mathbb{1}_t = \mathbb{1} - 2|t\rangle\langle t|$, construído a partir de $f_t(x)$ sin tener conocimiento explícito de $|t\rangle$
   - U = cualquier operador unitario tal que $\langle t|U|s\rangle \neq 0$, generalmente es H el operador Walsh-Hadamard.
2. Computar $|\psi\rangle = Q^k U|s\rangle$, es decir, iterar el operador Q, $k = \frac{\pi}{4}\sqrt{N}$ veces sobre el estado $U|s\rangle$.
3. Medir cada uno de los n valores de bit del estado $|\psi\rangle$.
4. Resultado: con alta probabilidad, el estado objetivo $|t\rangle$.

2.5.4. Funcionamiento del algoritmo de Grover

En el operador de amplificación de amplitud Q, los operadores $\mathbb{1}_s$ y $\mathbb{1}_t$ realizan inversiones de fase controladas. Específicamente $\mathbb{1}_s|x\rangle = -|x\rangle$ si y sólo si $x = s$. Igualmente, $\mathbb{1}_t|x\rangle = -|x\rangle$ si y sólo si $x = t$. Con estos operadores se puede construir un operador que aumenta la amplitud del eigen-estado objetivo.

Al expandir los términos de $-U\mathbb{1}_s U^\dagger \mathbb{1}_t$, se obtiene explícitamente el operador de amplificación de amplitud, dado por en la Ecuación 22.

$$Q = -\mathbb{1} + 2\left|t\right\rangle\left\langle t\right| + 2U\left|s\right\rangle\left\langle s\right|U^\dagger - 4U\left|s\right\rangle\left\langle s\right|U^\dagger\left|t\right\rangle\left\langle t\right| \tag{22}$$

En la Ecuación 23 se escribe en forma matricial la acción del operador Q sobre los estados $U\left|s\right\rangle$ y $\left|t\right\rangle$:

$$Q\begin{pmatrix} U\left|s\right\rangle \\ \left|t\right\rangle \end{pmatrix} = \begin{pmatrix} 1-4\left|\left\langle t|U|s\right\rangle\right|^2 & 2\left\langle t|U|s\right\rangle \\ -2\left\langle t|U|s\right\rangle^* & 1 \end{pmatrix}\begin{pmatrix} U\left|s\right\rangle \\ \left|t\right\rangle \end{pmatrix}. \tag{23}$$

Cuando el traslape entre $U\left|s\right\rangle$ y $\left|t\right\rangle$ es muy pequeño, es decir, cuando $u = \left\langle t|U|s\right\rangle << 1$, dichos estados son casi ortogonales uno con otro, de modo que Q se comporta como una compuerta de rotación de 1-qubit en el espacio que es generado por $U\left|s\right\rangle$ y $\left|t\right\rangle$. De modo que $\left|\left\langle t|U|s\right\rangle\right|^2 << \left|\left\langle t|U|s\right\rangle\right|$, es decir, $|u|^2 << |u|$ y la ecuación matricial de arriba se convierte en la Ecuación 24.

$$Q\begin{pmatrix} U\left|s\right\rangle \\ \left|t\right\rangle \end{pmatrix} \approx \begin{pmatrix} 1 & 2u \\ -2u^* & 0 \end{pmatrix}\begin{pmatrix} U\left|s\right\rangle \\ \left|t\right\rangle \end{pmatrix}$$

$$= \exp\begin{pmatrix} 0 & 2u \\ -2u^* & 0 \end{pmatrix}\begin{pmatrix} U\left|s\right\rangle \\ \left|t\right\rangle \end{pmatrix} \tag{24}$$

En la forma de matriz exponencial se puede calcular la $k-$ésima potencia de Q, finalmente dada por la Ecuación 25.

$$Q^k\begin{pmatrix} U\left|s\right\rangle \\ \left|t\right\rangle \end{pmatrix} = \exp\begin{pmatrix} 0 & 2ku \\ -2ku^* & 0 \end{pmatrix}\begin{pmatrix} U\left|s\right\rangle \\ \left|t\right\rangle \end{pmatrix}$$

$$Q^k\begin{pmatrix} U\left|s\right\rangle \\ \left|t\right\rangle \end{pmatrix} \approx \begin{pmatrix} \cos(2k|u|) & \frac{u}{|u|}\sin(2k|u|) \\ -\frac{u^*}{|u|}\sin(2k|u|) & \cos(2k|u|) \end{pmatrix}\begin{pmatrix} U\left|s\right\rangle \\ \left|t\right\rangle \end{pmatrix} \tag{25}$$

Esto implica que después de k iteraciones de Q, el traslape entre el estado $\left|t\right\rangle$ desco-

nocido y el estado amplificado $Q^k U|s\rangle$ está dado por la Ecuación 26:

$$\langle t|Q^k U|s\rangle \approx u\cos(2k|u|) + \frac{u}{|u|}\sin(2k|u|) \tag{26}$$

Aunque estas aproximaciones sólo son válidas cuando $u = \langle t|U|s\rangle << 1$, el cálculo ilustra la característica principal de la amplificación de amplitud. Con u pequeña y k modesta, $u\cos(2k|u|) \approx u$ y $\frac{u}{|u|}\sin(2k|u|) \approx 2ku \equiv 2k\langle t|U|s\rangle$. Entonces se puede aproximar el traslape como en la Ecuación 27:

$$\langle t|Q^k U|s\rangle \approx (1+2k)\langle t|U|s\rangle, \tag{27}$$

que implica un crecimiento lineal del traslape en el número de pasos de la amplificación de amplitud, k. De modo que, la probabilidad de obtener el estado objetivo en la medición, crece cuadráticamente en el número de pasos de amplificación de amplitud como se puede observar en la Ecuación 28:

$$p_{\text{éxito}}^{\text{QUANTUM}} \sim k^2 |\langle t|U|s\rangle|^2 \tag{28}$$

Hay que recordar que la probabilidad está dada por el cuadrado de la amplitud; de modo que cada incremento lineal en la amplitud implica un incremento cuadrático de la probabilidad.

2.5.5. Estimación del número de iteraciones requeridas por la Amplificación de Amplitud

Después de k iteraciones de amplificación de amplitud se tenía que el operador estaba dado por la Ecuación 29:

$$Q^k = \begin{pmatrix} \cos(2k|u|) & \frac{u}{|u|}\sin(2k|u|) \\ -\frac{u^*}{|u|}\sin(2k|u|) & \cos(2k|u|) \end{pmatrix} \tag{29}$$

que es casi la misma matriz que rota un vector en un ángulo θ, dicha matriz tiene la forma de la Expresión 30:

$$\begin{pmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{pmatrix} \tag{30}$$

Como $U|s\rangle$ y $|t\rangle$ son casi ortogonales inicialmente, se tiene que aplicar Q hasta rotar $U|s\rangle$ aproximadamente $\pi/2$ y alcanzar $|t\rangle$. Si en ese momento se midiera el registro, éste se encontrará en el estado objetivo $|t\rangle$ con probabilidad $O(1)$. Por tanto, el número de aplicaciones de Q requeridas para rotar $U|s\rangle$ hasta $|t\rangle$ estará dado por $2k|u| = \frac{\pi}{2}$, que implica $k = \frac{\pi}{4u}$. Al resolver para el valor real no nulo más pequeño de k tal que

$$u\cos(2k|u|) + \frac{u}{|u|}\sin(2k|u|) = 1$$

también implica que $k \to \frac{\pi}{4}\sqrt{N}$, es decir, las k iteraciones estarán dadas por la Ecuación 31.

$$k \approx \frac{\pi}{4}|\langle t|U|s\rangle|^{-1} = \frac{\pi}{4}\sqrt{N} \tag{31}$$

donde $N = 2^n$ es el número de elementos sobre los que se realiza la búsqueda. Es así, que el algoritmo de Grover es cuadráticamente más rápido que el algoritmo clásico para búsqueda no estructurada.

El número óptimo de iteraciones es $\frac{\pi}{4}\sqrt{N}$ aplicaciones de Q, ahí se tiene la amplitud máxima del estado $|t\rangle$. Algo delicado del algoritmo de Grover, y en general en algoritmos cuánticos de amplificación de amplitud, es que al sobrepasar la cantidad de iteraciones óptimas para alcanzar la máxima amplificación en $|t\rangle$, se decrementa la probabilidad de encontrar dicho estado; es decir, $U|s\rangle$ dejará de traslaparse casi perfectamente con $|t\rangle$ y se empezará a alejar conforme aumenta las iteraciones. Después de alejarse y casi volver a ser ortogonales (es decir, de volver al estado inicial) se repetirá un ciclo y en $\frac{\pi}{4}\sqrt{N}$ aplicaciones de Q la amplitud del estado $|t\rangle$ será máxima una vez más.

2.6. Complejidad cuántica

El estudio de las clases de complejidad cuánticas comenzó con David Deutsh en su artículo acerca de las máquinas de Turing cuánticas (Deutsch, 1985) donde presentó la idea del paralelismo cuántico. Hay que tomar en cuenta que el paralelismo cuántico permite evaluar ciertas funciones en todo su dominio, que puede ser de tamaño exponencial, en el tiempo que toma hacer sólo una evaluación de función clásicamente. Desafortunadamente, las leyes de la mecánica cuántica sólo permiten extraer la infor-

mación de un sólo camino computacional explícitamente dependiendo de una cierta probabilidad (véase la Figura 11). El problema es que aunque se puede calcular el valor de la función, para toda posible entrada de una sóla vez; cuando se lee la respuesta final de la cinta de una MTQ, sólo se puede obtener una de todas las posibles salidas. Una vez que se revela la respuesta, se pierde la información de todas las otras salidas sin posibilidad de rescatarlas.

En el modelo query se tiene que calcular una función $f(x_1, ..., x_N)$ de una entrada $(x_1, ..., x_N)$, con x_i accesible vía consultas a una caja negra, que dado i, genera x_i. La complejidad de un algoritmo se mide por la *query complexity* o complejidad de consulta en español: el número de veces que el algoritmo llama a la caja negra. Generalmente se analiza la complejidad de un algoritmo cuántico por medio de la *query complexity* al medir cuántas veces se llama a cierto oráculo u operador query, pues de esta forma se pueden probar cotas inferiores y es posible caracterizar la ventaja cuántica dentro de un factor *big-O* (Véase el Anexo para más información sobre la notación *Big-O*). El modelo query es muy interesante en el caso cuántico porque captura la mayoría de los algoritmos cuánticos conocidos. A continuación se enlistan algunos problemas que se pueden describir en este modelo:

- **Búsqueda no estructurada:** Dado un acceso de *caja negra* a $x_1, x_2, ..., x_N \in \{0, 1\}$, determinar si existe $i : x_i = 1$ (o encontrar tal i). La *query complexity* para resolver este problema es $O(\sqrt{N})$ tal como se calculó en la sección anterior.
- **Búsqueda de periodo:** Dado un acceso de caja negra a $x_1, ..., x_N \in [M]$, determinar el r más pequeño tal que $x_i = x_{i+r}$ para todo i (y $x_i \neq x_{i+q}$ para todo i y $q < r$), bajo la promesa de que tal r existe y es menor que $c\sqrt{N}$ para algún $c > 0$. Este problema es resoluble con $O(1)$ consultas cuánticas. Es el corazón del *Algoritmo de Shor* para factorización (Shor (1999)).
- **Distinción de elementos:** Dado un acceso de caja negra a $x_1, ..., x_N \in [M]$, determinar si existen $i, j : i \neq j$ tales que $x_i = x_j$. Se puede resolver en $\Theta(N^{2/3})$ consultas cuánticas.

El uso del modelo *query complexity* es necesario para el análisis de algoritmos en computación cuántica pues sin este modelo sería muy difícil comparar una solución

clásica con su contraparte cuántica. Tan difícil como hacer una separación de las clases entre *P* y *PSPACE* o *NP* de *BQP* (BQP siendo la clase de los problemas de decisión decidibles por una computadora cuántica en tiempo polinomial con probabilidad acotada a 1/3 sobre cualquier caso particular).

En el *mundo relativizado* los algoritmos tienen acceso a oráculos cuya estructura interna no se examina, en ese mundo se puede probar incondicionalmente que $BPP \neq BQP$; es decir, si ambas computadoras tienen acceso al mismo oráculo, existen ciertos problemas que las computadoras cuánticas pueden resolver exponencialmente más rápido que las clásicas. Cuando no se permite la existencia de oráculos se dice que se está en en el *mundo no relativizado*, y aquí no es posible separar las clases de complejidad; como cualquiera que se encuentre entre *P* y *PSPACE*. De modo que las comparaciones entre algoritmos cuánticos y clásicos es un problema abierto que llega hasta el mismo *P* vs *NP* y todas sus implicaciones.

2.6.1. El problema P vs NP

Es prudente describir a grandes rasgos el problema *P* vs *NP*; para estar conscientes de lo complicado que es realizar separaciones, contenciones estrictas o colapsos de unas clases en otras y motivar el uso de oráculos para comparar algoritmos cuánticos y clásicos.

- La clase de complejidad *P* es la clase de los problemas de decisión que pueden ser resueltos por una máquina de Turing determinista en tiempo polinomial.
- La clase de complejidad *NP* es la clase de los problemas de decisión que pueden ser resueltos por una máquina de Turing no determinista en tiempo polinomial. Alternativamente, la clase de complejidad *NP* es la clase de los problemas de decisión cuyas soluciones para los casos *SI* se pueden verificar en tiempo polinomial por una máquina de Turing determinista.

El problema *P* vs *NP* se pregunta si, dado que se puede verificar la solución a un problema eficientemente, ¿Es posible idear un algoritmo que encuentre dicha solución

eficientemente?, esto es, ¿Es P igual a NP? La gran mayoría de los computólogos y matemáticos creen que *P* y *NP* no son iguales. Sin embargo, la prueba ha escapado por décadas desde que se definió el problema; y, para aquel que resuelva *P* vs *NP* le espera una medalla Fields al cuello y un millón de dólares en la bolsa.

Lamentablemente, para la computación cuántica, encontrar que $P \neq NP$ no permitiría separar absolutamente a *BQP* de *BPP* (La versión probabilística clásica de BQ) o siquiera *BQP* de *NP*; aunque para esta última se puede realizar una separación relativizada. El que existan casos específicos de problemas *NP* que se puedan resolver por una computadora cuántica en tiempo polinomial tampoco dice nada acerca de las relaciones entre clases de complejidad. Si mostrar relaciones en complejidad clásica es difícil, agregar un modelo de cómputo de naturaleza distinta le añade aún más dificultad.

En resumen, cuando se discuta la complejidad en un algoritmo cuántico estaremos hablando de *query complexity*. No se puede cuantificar un *tiempo de ejecución* debido a la diferencia de naturalezas entre la computación cuántica y la clásica, el hecho de que existan primitivas en la primera sin análogo en la segunda hace sumamente difícil las separaciones en el mundo sin oráculos. En la descripción del algoritmo de Mateus para búsqueda de patrones se mencionará la *circuit complexity* para hablar sobre implementabilidad, donde la *circuit complexity* es la cantidad de compuertas requeridas en la construcción de un algoritmo (u oráculo).

Capítulo 3. El problema de la búsqueda de patrones y jumbled matching

En este capítulo se hace una revisión rápida al problema de la búsqueda de patrones, se describe a detalle el problema de *emparejamiento entremezclado* (Se usará *emparejamiento entremezclado*, o el término en inglés, *jumbled matching* indistintamente) y sus soluciones clásicas. Finalmente, se presenta la prueba de dureza que establece que al utilizar un índice se puede volver constante el tiempo de consulta a costa de usar espacio de memoria cuadrático; o bien, consultar en tiempo lineal sin recurrir al uso de un índice. Todo esto con el objetivo de motivar la exploración del paradigma de la computación cuántica para construir un algoritmo que resuelva el problema de jumbled matching. Con ello, estaremos listospara presentar la solución desarrollada en este trabajo de tesis. Las primeras secciones de este capítulo serán de mucha utilidad para la presentación del algoritmo cuántico para pattern matching.

Una de las tareas de mayor interés en las ciencias de la computación es la búsqueda de patrones, donde se desea encontrar cierta información (el patrón) de entre un conjunto grande de datos. En este trabajo, los patrones son cadenas generadas a partir de un cierto alfabeto, contenidas en una base de datos que se nombra como *texto* y que es en realidad una cadena larga. Es posible encontrar el problema de *pattern matching* en infinidad de aplicaciones: desde buscar una palabra dentro de un documento de texto o en una búsqueda de Google, hasta hacer detección de plagio o detección en espectrometría de masa.

3.0.1. Definición del problema de búsqueda de patrones

Pattern Matching Dado un texto T de longitud N y un patrón p de longitud m, ambos construidos a partir de un alfabeto Σ; encontrar todas las posiciones i tales que $T[i, \ldots, i+m-1] = p$.

La definición anterior es para el problema de la búsqueda exacta de patrones. Sin embargo, en casos prácticos donde los datos pueden mutar en el tiempo o son guardados en sistemas sensibles a fallas, el problema de la búsqueda de patrones más

cercanos es uno mucho más relevante, ya que algunas veces sólo existen aproximaciones al patrón y por tanto encontrar coincidencias exactas es imposible. Para este caso también se define el problema:

Pattern Matching Dado un texto T de longitud N y un patrón p de longitud m, ambos construidos a partir de un alfabeto Σ; se desea saber si p ocurre en T. En particular, se quiere encontrar la posición $i \in \{1, \ldots, N\}$ donde un cierto símbolo de p ocurre en T.

3.0.2. Algoritmo de Knuth-Morris-Pratt

Se presentará la solución ingenua al problema de pattern matching y, después, el algoritmo de Knuth-Morris-Pratt que hace eficiente la aproximación por ventanas deslizantes.

El algoritmo ingenuo para pattern matching es usar una ventana deslizante de tamaño m que en una primera iteración busca la coincidencia al comparar cada elemento del patrón p con $T[1, 2, \ldots, m]$, si no es el caso la ventana avanza hacia la derecha de T y repite. El tiempo de ejecución de este algoritmo simple es, en peor caso $O(Nm)$.

El algoritmo de Knuth-Morris-Pratt es el estado del arte (junto con el algoritmo de Boyer-Moore) en la búsqueda de patrones en cadenas; algunas variaciones recientes mejoran los tiempos promedios pero siguen teniendo los mismos tiempos en peor caso. El algoritmo de Knuth-Morris-Pratt (KMP) utiliza una técnica de ventana deslizante que evita las comparaciones extra de subpatrones en T que contengan los primeros elementos de p. Un ejemplo de este problema es al buscar *aaab* en el texto *baaaab*; en la segunda y tercera iteración se tomarían ciclos innecesarios al comparar las primeras tres *a* de *aaab*. De cierta forma, se puede pensar en el algoritmo KMP como un algoritmo de ventana deslizante con memoria. El algoritmo precalcula una tabla llamada *tabla de fallos* que guiará a la ventana deslizante en los saltos que tendrá que hacer a la derecha para evitar las comparaciones de los elementos que son coincidentes. La tabla verifica si en el patrón existen repeticiones de prefijos, y cuando es el caso agre-

gará índices que apuntarán hacia la primera aparición de los prefijos, en dicho caso, la ventana no avanzará en una sola posición en el texto T, sino que saltará en la misma cantidad de elementos donde empieza la repetición de prefijos. Si en un momento no se llega al final de la tabla de fallos significa que aún no se encuentra el patrón p y la ventana deslizante avanzará hacia la derecha en un elemento, y se repite el proceso.

3.1. El problema de Jumbled Pattern Matching

Se puede cuantificar la similaridad entre dos distintas cadenas empleando distintas métricas. Por ejemplo, la distancia de edición o *distancia de Levenshtein*, que define una métrica basada en el número de inserciones, borrados, o sustituciones requeridas para convertir una cadena en otra; esta métrica puede ser aplicada sin importar si las cadenas a analizar tienen distintas longitudes. Si en cambio se tienen cadenas con la misma longitud es posible aplicar la distancia de Hamming, que mide el número de posiciones en las que dos cadenas difieren. Existe otra forma de cuantificar la semejanza entre cadenas de la misma longitud: la distancia *jumbled* o de revoltura. Más que una distancia, es una semi-métrica discreta que verifica si dos cadenas son construidas con los mismos caracteres y la misma cantidad de estos. La salida de la comparación de dos cadenas bajo esta semimétrica será 0 para un emparejamiento entremezclado o *jumbled match*, y en caso contrario, será 1 y no se habrá dado un *jumbled match*. Jumbled pattern matching (JPM) surge en problemas de biocomputación relacionados con análisis de patrones, especialmente en espectrometría de masa para interpretación de datos, descubrimientos de SNP, alineamiento de secuencias, y aplicaciones de biosecuenciación en general. Donde no es necesario obtener coincidencias exactas; en cambio, se requiere encontrar una versión permutada de subcadenas consulta; de este modo, la distancia *jumbled* es la métrica apropiada para el problema JPM.

3.1.1. Planteamiento del problema

Sea $\Sigma = \{a_1, ..., a_{|\Sigma|}\}$ un alfabeto finito ordenado, y una cadena $s \in \Sigma^*$. Un vector de Parikh cuenta la frecuencia de los elementos que componen una cadena, es decir, el vector de Parikh de alguna cadena s tiene la forma $p(s) = \{p_1, p_2, ..., p_{|\Sigma|}\}$, donde $p_i = |\{j | s_j = a_i\}|$ es el número de ocurrencias del i—ésimo elemento de Σ. Dada una

cadena T de tamaño N y una subcadena q de tamaño m, ambas construídas con el mismo alfabeto Σ, y $m \leq N$, se dice que una ocurrencia del vector de Parikh $p(q)$ en la cadena T toma lugar cuando una subcadena en T tiene la misma multiplicidad de caracteres de q; esto es, tienen el mismo vector de Parikh.

Jumbled Pattern Matching. Sea Σ un alfabeto, T un texto de tamaño N y q una subcadena en T. Dado un vector de Parikh consulta p_q, encontrar todas las subcadenas $s \in T$ tal que $p_q = p_s$.

Ejemplo

Sea $s = abcdebabdebaabb$ una cadena sobre el alfabeto $\sum = \{a, b, c, d, e\}$ y $q = \{1, 2, 0, 0, 0\}$ el vector de Parikh de la cadena abb.

Al buscar q con algún algoritmo obtendremos los índices $i = 5$ e $i = 13$ de las subcadenas $t_1 = bab$ y $t_2 = abb$ respectivamente ya que sus vectores de Parikh son:

$$p(t_1) = p(t_2) = \{1, 2, 0, 0, 0\} = q$$

3.1.2. Soluciones clásicas

En aplicaciones reales se considera la longitud de la subcadena q, como constante; por supuesto, $|q| << |T|$. Se puede resolver trivialmente el JPM para un sólo patrón en $O(N)$ tiempo con un algoritmo simple de ventana deslizante, buscando exhaustivamente a lo largo de T la subcadena con el mismo vector de Parikh. Para acelerar el tiempo de consulta de JPM se puede preprocesar T para construir un índice; a este enfoque se le llama *jumbled indexing* (JI). La técnica JI busca una construcción eficiente de un índice donde se tienen muchas cadenas por un lado y una consulta por el otro. En la búsqueda indexada, por ejemplo, usando la métrica de Hamming o de Levenshtein, el objetivo es encontrar aquellas cadenas con distancia mínima a la consulta; mientras que, en JI, el objetivo es encontrar todas las cadenas que hagan jumbled match con la consulta. En la búsqueda indexada se construye el índice antes de las consultas; después de eso, la búsqueda de las coincidencias tomará tiempo constante.

Este problema ha sido complicado de atacar, los investigadores han intentado mejorar la solución por casi cuarenta años al buscar un índice universal; sin embargo, hasta ahora no ha sido posible. Un caso particular consiste en tener un texto simple T en el que todas las subcadenas de tamaño m se consideran, donde m es el tamaño de la consulta, y se quiere encontrar cuál coincide con la consulta. Existen dos soluciones triviales:

- *Sin índice, en línea*, de solución lineal con un algoritmo simple de ventana deslizante.
- *Con un índice*, usando tiempo y espacio cuadrático para construir el índice de todas las subcadenas; es decir, construir diccionarios para tamaños $2, 3, \ldots, N$, cada uno representando un vector de Parikh distinto. El tiempo de consulta es constante.

Hasta antes de 2014 se desconocía si era posible construir índices en menos de orden cuadrático, o bien, buscar en tiempo sublineal sin un índice. En Amir *et al.* (2014) se probó mediante dureza condicional que no es posible. Este resultado es central para nuestro trabajo, ya que motiva la exploración de soluciones alternativas en el paradigma de la computación cuántica.

3.2. Prueba de dureza condicional

En Burcsi *et al.* (2010); Moosa y Rahman (2010, 2012) se pueden ver ejemplos de algoritmos para JI donde el tiempo de ejecución del pre-procesamiento o de consulta tiene factores polilogarítmicos sobre los algoritmos ingenuos. Estos han dado pistas para la imposibilidad de la aceleración de JI. En un sentido, los algoritmos ingenuos son lo mejor que se puede hacer para JI. Probar cotas inferiores para un problema específico es una tarea difícil, especialmente cuando el problema tiene complejidad polinomial. Recientemente, las técnicas de dureza condicional han dado a los investigadores una nueva herramienta. Como en las reducciones NP-hard, y permitiendo cuidadosamente sólo transformaciones de tiempo lineal, ellos escogen un problema más difícil para hacer la reducción. A continuación se explican algunas primitivas difíciles comúnmente usadas en este paradigma, y los tiempos de ejecución que se conjeturan

como los mínimos posibles para cualquier algoritmo que resuelva cada problema.

- *APSP*, o *camino más corto entre todo par de vértices de un grafo* (All-Pairs Shortest Path en inglés), cuya solución tiene un tiempo de ejecución de $O(n^3)$.
- *3-SAT* o *3-Satisfacibilidad booleana* (Boolean satisfiability en inglés), donde se desea saber si existe una asignación de variables que satisfaga una fórmula Booleana de 3 cláusulas, la complejidad para cualquier algoritmo que lo resuelva no es mejor que exponencial en peor caso; a esta suposición se le llama la *Hipótesis de Tiempo Exponencial (ETH)*, existe una suposición equivalente para $k-SAT$, llamada *Hipótesis Fuerte de Tiempo Exponencial (SETH)*.
- *3SUM*, donde dado un conjunto de N números reales nos preguntamos si existen tres de ellos tales que sumen cero, el tiempo de ejecución esperado para resolver este problema es $O(n^2)$.
- *OV* o *Vectores ortogonales* (Orthogonal Vectors en inglés). En este problema se dan dos conjuntos U y V de n vectores cada uno, sobre el conjunto $\{0,1\}^d$ donde $d = \omega(\log n)$, y queremos determinar si existen $u \in U$ y $v \in V$ tales que $\Sigma_{i=1}^{d} u_i \cdot v_i = 0$. No se conoce ningún algoritmo con tiempo de ejecución mejor que $O(n^2)$.
- *HS* o *Núcleo de Cobertura de Universo Pequeño* (Small Universe Hitting Set en inglés), que es una version del problema *OV*. Dado dos conjuntos U y V de vectores sobre $\{0,1\}^d$ para $d = \omega(\log n)$, determinar si existe un $u \in U$ tal que para todo $v \in V$, $\Sigma_{i=1}^{d} u_i \cdot v_i > 0$. Se conjetura que no existen algoritmos que resuelvan *HS* en tiempo subcuadrático.
- *Emparejamiento de Triángulos* (Matching Triangles en inglés), Dado un grafo G de n vértices coloreados, ¿Existe una tripleta de colores distintos a, b, c tales que hay al menos Δ triángulos (x, y, z) en G en los que x tiene color a, y tiene color b, y z tiene color c? Matching Triangles se puede resolver en $O(n^3)$.
- *Colección de Triángulos* (Triangle Collection en inglés), Dado un grafo $G = (V, E)$ de n vértices coloreados, ¿Existe una tripleta de colores distintos a, b, c tales que no hay triángulos en G con esos colores? Al igual que el problema anterior, el tiempo de ejecución de este problema es de $O(n^3)$.

Después de usar reducciones hacia estas primitivas, se pueden probar cotas inferiores del problema original mediante sus conjeturas de dureza. (Ver Williams (2015) para una discusión más detallada de esta técnica).

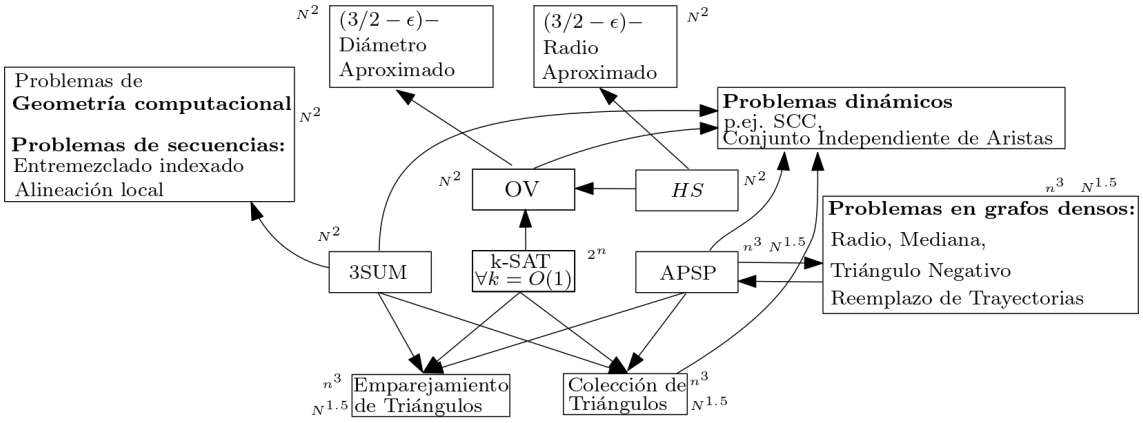

Figura 12. Algunas implicaciones de las principales conjeturas de dureza. Una flecha desde un problema A hacia un problema B, donde A tiene $a(n)$ al lado, B tiene $b(n)$ al lado, implica que el problema A es (a, b)-reducible a B.

Para entender cómo se han desarrollado las pruebas de dureza condicional, se resumen grosso modo cómo se prueba que un cierto problema de decisión se encuentra contenido en la clase de los problemas NP-Completo. El esquema para el desarrollo de estas pruebas es mediante el diseño de reducciones polinomiales:

Sea D un poblema de decisión expresado como un lenguaje formal L, para probar que $L \in NP-Completo$:

1. Se muestra que L está en NP
2. Se elije un L' que se conoce en NP-Completo para mostrar que L' reduce a L:
   (a) Diseñar una transformación polinomial f.
   (b) Mostrar que f se puede calcular en tiempo determinístico polinomial.
   (c) Mostrar que $w \in L' \iff f(w) \in L$.

Este esquema para probar contenciones de problemas en la clase $NP-Completo$ ha inspirado a los investigadores para desarrollar técnicas que expliquen por qué existen

problemas con tiempo de ejecución $O(n^k)$ que no se han mejorado en décadas de investigación; probablemente, la respuesta es porque no existen algoritmos que los resuelvan en menos de $\Omega(n^{k-o(1)})$ tiempo de ejecución. Probar directamente estas cotas se ha mantenido como una tarea muy difícil; sin embargo, imitando el esquema anterior se han encontrado cotas inferiores para algunos de estos problemas de la siguiente forma:

1. Se escogen unos pocos problemas clave de los cuales se conjetura que requieren $T(n)$ tiempo para resolverse.
2. Se diseñan reducciones especiales llamadas *fine-grained reductions* o reducciones muy finas, de complejidad lineal o menor.
3. Mediante estas reducciones y con las conjeturas o *suposiciones de dureza* se prueban cotas inferiores para muchos problemas contenidos en P.

A este tipo de dureza, o dificultad de un problema se le llama *dureza condicional* pues está condicionada a la supuesta imposibilidad de encontrar soluciones más rápidas para los problemas clave; de modo que, en lugar de probar cotas inferiores directamente sobre el problema, se hace una reducción y de forma indirecta se establecen cotas inferiores en los tiempos de ejecución.

3.2.1. Reducciones muy finas

Para aplicar el esquema anterior se definen formalmente las *reducciones muy finas*, en las que para problemas A y B y tiempos de ejecución $a(n)$ y $b(n)$, implican que un algoritmo para B de tiempo $O(b(n)^{1-\epsilon})$ para una constante $\epsilon > 0$ implica un algoritmo para A de tiempo $O(a(n)^{1-\delta})$ para una constante δ.

Definición 7 *(Reducción muy fina)*

Sea $a(n)$ y $b(n)$ funciones no decrecientes de n. Un problema A es (a, b)-reducible a un problema B, denotado como $A \leq_{a,b} B$, si para cualquier $\epsilon > 0$, existe un $\delta > 0$, un algoritmo F con acceso a un oráculo a B, una constante d, y para cada $n \geq 1$ un entero $k(n)$, tal que para cada n, el algoritmo F toma cualquier caso específico de A de tamaño n y

- *F corre a lo más en* $d \cdot (a(n))^{1-\delta}$ *unidades de tiempo,*
- *F produce a lo más* $k(n)$ *casos específicos de B de forma adaptativa, esto es, el* $j-$*ésimo caso específico* B_j *es una función de* $\{(B_i, a_i)\}_{1 \leq i < j}$ *donde* B_i *es el* $i-$*ésimo caso específico y* a_i *es la respuesta del oráculo para B sobre el caso específico* B_i*, y*
- *los tamaños* n_i *de los casos específicos* B_i *para cualquier elección de oráculo que responde* a_i *obedecen la desigualdad* $\Sigma_{i=1}^{k(n)}(b(n_i))^{1-\epsilon} \leq d \cdot (a(n))^{1-\delta}$

Una consecuencia inmediata de estas reducciones es que las mejoras sobre $b(n)$ para B implican mejoras sobre $a(n)$ para A. Formalmente, si existe un algoritmo para B de tiempo de ejecución $c \cdot N^{1-\epsilon}$ sobre casos específicos de tamaño N, entonces componer el algoritmo con una $(a, b)-$reducción muy fina desde A a B produce un algoritmo para A con tiempo de ejecución dado por la Ecuación 32:

$$d \cdot (a(n))^{1-\delta} + \Sigma_{i=1}^{k(n)} c \cdot (b(n_i))^{1-\epsilon} \leq d \cdot (c+1) \cdot (a(n))^{1-\delta}, \tag{32}$$

donde el primer sumando es para correr el algoritmo de la reducción F, y el segundo sumando es el tiempo de ejecución para B sobre los casos específicos B_i. Como c y d son constantes, el tiempo de ejecución para A es $O(a(n)^{1-\delta})$

Es fundamental notar que las reducciones necesarias para mostrar cotas inferiores en los problemas con soluciones de tiempo de ejecución polinomial deben ser lineales o menores. De modo que el tiempo de ejecución total de la reducción más las soluciones para todos los casos específicos B_i no debe ser mayor al tiempo de ejecución del caso específico del problema A desde el que se desea hacer la reducción.

3.2.2. Convolution-3SUM y conjetura de dureza

Ahora que se definieron las reducciones muy finas y se revisó el mecanismo con el cuál estas reducciones ayudan a encontrar cotas inferiores de problemas particulares, se presentarán otras definiciones importantes para el desarrollo de la prueba

de dureza para *Jumbled Indexing*. La primera de estas definiciones es el problema 3SUM, a partir del cual se conjetura la suposición de dureza de tiempo cuadrático para cualquier algoritmo que lo resuelva:

Definición 8 (El problema 3SUM) *Dado un conjunto* $S = x_1, x_2, \ldots, x_n$*, responder* Si*, si existen tres distintos* i, j *y* k *tal que* $x_i + x_j = x_k$*. Responder* No*, en otro caso.*

Aplicar directamente 3SUM en problemas de combinatoria hace que las transformaciones sean largas y poco claras. Para superar este problema se define una forma restringida de 3SUM, llamada *Convolution-3SUM* que genera reducciones más simples, pero de la misma dificultad; es decir, que la conjetura de dureza para 3SUM se mantiene para convolution-3SUM (Patrascu (2010)).

Definición 9 (El problema Convolution-3SUM) *Dado un conjunto* $S = x_1, x_2, \ldots, x_n$*, responder* Si*, si existen distintos* i *y* j *tal que* $x_i + x_j = x_{i+j}$*. Responder* No*, en otro caso.*

Se puede escribir una definición equivalente pero más conveniente para la demostración:

Dado un conjunto $S = x_1, x_2, \ldots, x_n$*, responder* Si *si existen distintos* i *y* j *tal que* $x_i - x_j = x_{i-j}$*. Responder* No*, en otro caso.*

La motivación para realizar la transformación de JI hacia convolution-3SUM recae en la suposición de que convolution-3SUM (O 3SUM, recordar que ambos problemas son de la misma dificultad) requiere un mínimo tiempo de ejecución. A la conjetura de la existencia de esta cota inferior se le llama *suposición de dureza*; existen conjeturas parecidas para APSP, K-SAT y otros problemas computacionales listados en subsecciones anteriores.

La conjetura de dureza supone el modelo *Word RAM* con longitud de palabras $O(\log n)$.

Suposición de dureza de 3SUM El tiempo esperado de cualquier algoritmo para resolver Convolution-3SUM es $n^{2-o(1)}$ para verificar si un conjunto $\{x_1, x_2, ..., x_n\}$ en $\{-n^2, ..., n^2\}$ contiene un par x_i y x_j tal que $x_i - x_j = x_{i-j}$.

En el problema original de 3SUM no existe una cota en los valores del conjunto $\{x_1, x_2, ..., x_n\}$ ya que estos son números reales; sin embargo, se puede utilizar un hash perfecto para mapear ese rango de números al rango $\{-n^3, ..., n^3\}$ (Patrascu (2010)). Como se tratará con la versión convolution-3SUM, donde sólo hay n^2 tripletas ($x_i - x_j = x_{i+j}$), la reducción que inicialmente se puede hacer desde un dominio más largo $\{-u, ..., u\}$ hacia el dominio $\{-n^3, ..., n^3\}$ y aún así mantener la cota inferior, puede adaptarse para Convolution-3SUM de modo que el nuevo dominio sea $\{-n^2, ..., n^2\}$; de nuevo, manteniendo la dureza siempre que $u \geq n^2$. Una suposición adicional es que *convolution-3SUM* para entradas que pertenezcan a un dominio $\{-n, ..., n\}$ es tan difícil como el caso general. A esta última conjetura se le llama *Conjetura fuerte para la dureza de* 3*SUM*.

3.2.3. Descripción general de la demostración

En una descripción de alto nivel, la demostración para la cota inferior de jumbled indexing se basa en realizar una reducción muy fina desde el problema *Convolution-3SUM* hacia *JI*. Convolution-3SUM es tan difícil como 3*SUM* pero es una forma más conveniente para problemas de combinatoria (Patrascu (2010)). Aplicando una función hash que usa módulo sobre un conjunto de primos, una entrada de Convolution-3SUM es transformada a JI; los valores más pequeños resultantes del proceso de hashing son convertidos a una cadena larga. Finalmente, n consultas se generan de forma tal que cada una representa un elemento de la entrada de Convolution-3SUM.

3.2.4. Detalles de la construcción

Se construye una cadena que es la entrada para la instancia de JI que resulta de la reducción. El primer paso es tomar un conjunto de primos distintos de aproximadamente el mismo tamaño $\{p_1, ..., p_k\}$ para algún k fijo, tal que su producto $p_1 \cdots p_k > n^2$, (Se puede pedir que $p_1 \cdots p_k > n$ para el caso de la conjetura fuerte para

la dureza de 3SUM). La cadena de entrada S para el caso específico de JI se genera a partir de esos números primos, además, un conjunto $\{Q_1, \ldots, Q_n\}$ de n consultas se genera en la reducción. Directamente de las propiedades del módulo podemos derivar un lema útil para el proceso de reducción:

Lema 3.1 *Sean $\{p_1, \ldots, p_k\}$ primos distintos tales que $p_1 \cdots p_k > u$ (Con $u = n^2$ o $u = n$ dependiendo de la suposición de dureza). Sea $i > j$. Entonces $x_i - x_j = x_{i-j} \iff \forall r : (x_i - x_j) \mod p_r = x_{i-j} \mod p_r \iff \forall r : (x_i \mod p_r) - (x_j \mod p_r) \in \begin{cases} (x_{i-j} \mod p_r) \\ (x_{i-j} \mod p_r) - p_r \end{cases}$*

Demostración

$\Rightarrow$)

Como $x_i - x_j = x_{i-j}$, se cumple que $(x_i - x_j) \mod n = x_{i-j} \mod n$ para cualquier $n \in \mathbb{N}$

en particular, para todo $p_r \in \{p_1, \ldots, p_k\}$

es decir, $\forall r : (x_i - x_j) \mod p_r = (x_{i-j}) \mod p_r$

$\Leftarrow$)

Sea $a = x_i - x_j$ y $b = x_{i-j}$,

Al considerar la definición de mód , $a \equiv b \mod n \iff a - b$ es divisible por n.

$\forall r : a \mod p_r = b \mod p_r$ implica que $(a - b)$ es divisible por $p_1, p_2, \ldots, p_k$

Caso 1. $a - b = 0$

$a - b$ es divisible por $p_1 \cdot p_2 \cdots p_k$ y se cumple que $|a - b| < p_1 \cdot p_2 \cdots p_k$

Caso 2. $a - b \neq 0$

Sin pérdida de generalidad se supone $a - b > 0$, $a - b$ es divisible por $p_1 \cdot p_2 \cdots p_k$, por tanto $a - b \geq p_1 \cdot p_2 \cdots p_k$, pero $|a - b| \leq u < p_1 \cdot p_2 \cdots p_k$ por hipótesis. Por tanto, el caso dos no ocurre.

Cadena de entrada para JI Tomando la entrada $\{x_1, \ldots, x_n\}$ de Convolution-3SUM se va a generar la cadena de entrada S que se preprocesará. Como se dijo anteriormente, se crea un caracter a_j para cada primo que se escoge, y para cada x_i se forma

la subcadena de la Ecuación 33:

$$S_i = a_1^{EXP(i,1)} a_2^{EXP(i,2)} \cdots a_k^{EXP(i,k)}, \tag{33}$$

donde $EXP(i,j)$ está dado por la Ecuación 34:

$$EXP(i,j) = (x_{i+1} \mod p_j) - (x_i \mod p_j) \tag{34}$$

y a^r representa a repetida r veces.

El exponente anterior no puede ser negativo, pero es fácil arreglar este problema al hacer $D = \max_{i=1}^{k} p_i$ y cambiando $EXP(i,j) = (x_{i+1} \mod p_j) - (x_i \mod p_j) + D$; por tanto, tomamos la expresión (34) por simplicidad.
Finalmente, con las subcadenas generadas en el proceso anterior, se construye la cadena larga de entrada S, dada por la Ecuación 35:

$$S = \$\#S_1\#\$\#S_2\#\$\#\cdots\#\$\#S_{n-1}\#\$, \tag{35}$$

donde # y $ serán caracteres especiales.

Después de la reducción, el tamaño del alfabeto para JI es $|\Sigma| = k + 2$, donde cada caracter corresponde a cada primo que escogimos, más los caracteres separadores especiales. El siguiente lema se puede demostrar mediante la estructura de las subcadenas en S:

Lema 3.2 *Sea* $R_{(j,i)} = \$\#S_j\#\$\#...\#\$\#S_{i-1}\#\$$ *una subcadena de* S. *Para cada* a_l *hay exactamente* $(x_i \mod p_l) - (x_j \mod p_l)$ *ocurrencias en* $R_{(j,i)}$.

Demostración

El caracter a_l tiene $EXPj, l$ ocurrencias en S_j, $EXP(j+1, l)$ ocurrencias en $S_{j+1}, ...,$ $EXP(i-1, l)$ ocurrencias en S_{i-1}. Por tanto, se tienen $\Sigma_{d=j}^{i-1} EXP(d, l)$ ocurrencias de a_l en $R(j, i)$. Entonces $\Sigma_{d=j}^{i-1} EXP(d, l) = \Sigma_{d=j}^{i-1}(x_{d+1} \mod p_l) - (x_d \mod p_l)$, por propiedad telescópica, se obtiene $(x_i \mod p_l) - (x_j \mod p_l)$.

Directamente de los Lemas 3.1 y 3.2 se puede ver que:

Corolario 1 *Existe una solución $x_i - x_j = x_{i-j}$ para Convolution-3SUM si y sólo si el número de coincidencias de cada caracter a_l en $R_{(j,i)}$ está en el rango dado por la Ecuación 36:*

$$\{(x_{i-j} \mod p_l), (x_{i-j} \mod p_l) - p_l\} \tag{36}$$

Consultas JI Se necesitan generar un conjunto $\psi_1, \ldots, \psi_n$ de n consultas para el caso específico de JI de forma tal que un ψ_L particular representa un elemento particular x_L de la entrada para Convolutional-3SUM. La consulta ψ_L simula una consulta sobre los datos de la convolution-3SUM al preguntar si existen i y j que cumplan los siguientes criterios:

(a) $x_L = x_i - x_j$, y

(b) $L = i - j$.

(c) Del Lema 3.2, es necesario que cualquier subcadena que hace jumbled match con una consulta ψ debe ser de la forma $R_{j,i}$.

Claramente, se obtiene una solución para Convolution-3SUM al responder cada ψ_L pues cada consulta generada ψ_L imita una consulta en convolution-3SUM.

El requisito (c) se satisface al usar los separadores especiales; mientras que (b) significa que cada $R_{(i,j)}$ tiene que estar formado con exactamente L sub-cadenas S_h. Se puede forzar ambas condiciones mediante la siguiente observación:

Observación 1 *Cualquier subcadena de S que hace jumbled match con la consulta ψ, donde ψ tiene $L + 1$ elementos \$ y $2L$ elementos #, debe ser de la forma $R_{(i,j)}$ y satisfacer $L = i - j$.*

Satisfacer (a) significa encontrar las subcadenas $R_{(j,i)}$ tales que a_l en el Corolario 1 coincide con el número de ocurrencias para alguna de las dos posibilidades, como

ambas no se pueden verificar por una sola consulta de JI, se parte a ψ_L en 2^k consultas; ahora, es posible probar cada 2^k diferentes diferentes igualdades, proporcionando una respuesta completa para el problema Convolution-3SUM. Así que, al aplicar esta reducción para el preprocesamiento y después al responder las $2^k n$ consultas, se produce una solución para Convolution-3SUM. En resumen, se puede juntar todas estas ideas en el siguiente teorema para texto de tamaño s y alfabetos de tamaño $r \geq 5$:

Teorema 3.1 *Considérese el problema JI con texto de tamaño s y alfabeto de tamaño* $r \geq 5$*. Entonces, bajo la suposición de dureza 3SUM, una de las siguientes opciones se mantienen para cualquier* $\epsilon > 0$*:*

1. *El tiempo de preprocesamiento es* $\Omega(s^{2-\frac{4}{r}-\epsilon})$ *o*
2. *El tiempo de consulta es* $\Omega(s^{1-\frac{2}{r}-\epsilon})$

3.3. Búsqueda cuántica de patrones

En esta sección se describe un algoritmo cuántico para la búsqueda de patrones desarrollado por Mateus y Omar (2005), este algoritmo permite buscar cualquier cantidad de distintos patrones en una cadena dada, requiriendo una función query por símbolo del alfabeto. El algoritmo regresa la posición de la subcadena más cercana a un cierto patron de tamaño M con probabilidad no despreciable en $O(\sqrt{N})$ tiempo, donde N es el tamaño de la cadena larga.

En Hariharan y Vinay (2003) y Mateus y Omar (2005), los autores muestran algoritmos cuánticos para el problema de búsqueda de patrones. En Hariharan y Vinay (2003) los autores usan funciones query particulares dependientes del patrón buscado, mientras que en Mateus y Omar (2005) las funciones query son independientes del patrón. Desde un punto de vista de complejidad, las funciones query se pueden preparar previamente haciendo coincidir el número de oráculos con el tamaño del alfabeto. En aplicacaciones reales de búsqueda de patrones ésta puede ser una suposición razonable: por ejemplo, al buscar subcadenas en patrones de ADN, donde el tamaño del alfabeto es constante. A continuación se discute el problema resuelto en Hariharan y Vinay (2003) y Mateus y Omar (2005).

Definición 10 ***Distancia de Hamming***

Sean dos cadenas $u, v \in \Sigma$ del mismo tamaño, se define la distancia de Hamming entre u y v como el número de diferencias entre ellas.

Definición 11 ***Búsqueda de patrones con a lo más k diferencias***

Dadas $w, p \in \Sigma^*$ cadenas de tamaños $|w| = N$ y $|p| = M$, con $N >> M$, tomar un parámetro adicional k. Obtener todos los i, $1 \leq i \leq (N-M+1)$ para los que la distancia de Hamming entre $w_i w_{i+1}, \ldots, w_{i+m-1}$ y $p_1 p_2, \ldots, p_m$ es menor o igual a k.

Definición 12 ***Búsqueda de patrones más cercanos***

Dado el problema de búsqueda de patrones con a lo más k diferencias, devolver el patrón con la k más pequeña.

3.3.1. Preparación

El estado inicial del sistema captura una analogía del algoritmo clásico de ventana deslizante, en el que se inspecciona la cadena larga con una ventana de tamaño M. Se requiere un estado donde el segundo símbolo de p ocurra después del primero, el tercero después del segundo, etc. La preparación apropiada que satisface este requisito es como en la Ecuación 37:

$$|\psi_0\rangle = \frac{1}{\sqrt{N-M+1}} \sum_{k=1}^{N-M+1} |k, k+1, \ldots, k+M-1\rangle \tag{37}$$

Hay que notar que se empieza en un espacio de estados $H^{\otimes M}$, y justo después de la preparación, el espacio de estados se reduce a un subespacio de dimensión $N-M+1$.

En la Ecuación 38 se define una función query que usa la rutina de Grover:

$$f_\sigma(i) = \begin{cases} 1 & \text{si la } i-\text{ésima letra de } w \text{ es } \sigma \\ 0 & \text{de otra forma.} \end{cases} \tag{38}$$

En este enfoque, se necesitan operadores query, uno por cada elemento del alfabeto; de modo que, el cambio de fase de la subrutina de Grover la realiza la transformación

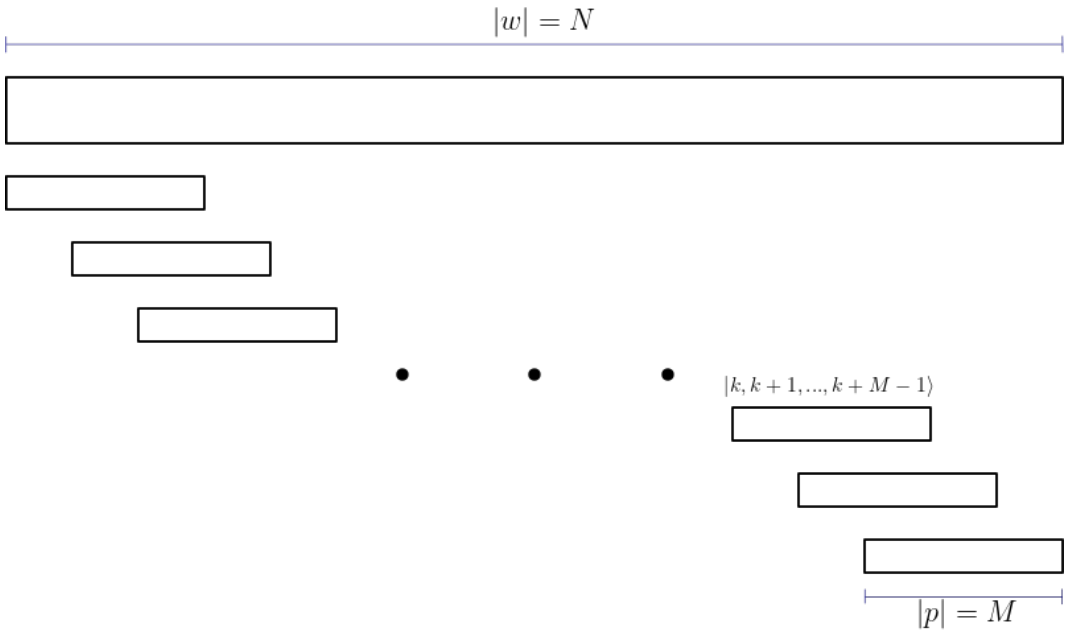

Figura 13. Representación gráfica de la preparación que emula al algoritmo clásico de ventana deslizante, donde el segundo símbolo de *p* ocurre después del primero, el tercero después del segundo, etc.

unitaria de la Ecuación 39:

$$U_\sigma |k\rangle = (-1)^{f_\sigma(k)} |k\rangle , \tag{39}$$

mientras que el operador de amplificación de amplitud será igual que en la descripción que se hizo del iterador de Grover en la sección 2.5.3.

3.3.2. Procesamiento

Cada símbolo del patrón, elegido de forma aleatoria, y su operador query correspondiente se aplican en la posición apropiada. Por tanto, en promedio, una posición con una coincidencia parcial; por ejemplo, *M'* de *M* coincidencias de símbolos individuales, se aplica el operador query $\frac{M'}{M}$ veces. Cada coincidencia produce una amplificación, resultando en una probabilidad más alta de detección al realizar la medición. Hay que notar que para una cierta cadena podrían haber coincidencias completas y parciales en el texto, conduciendo a amplificaciones de amplitud más grandes o más pequeñas en diferentes posiciones (Ver figura 14). Si $N >> M$, como en aplicaciones reales, muestrear alrededor de $\sqrt{N}$ veces los *M* elementos del patrón produce una amplificación significativa para ser medida con alta probabilidad.

Finalmente, se obtiene la posición de la coincidencia más cercana a p al realizar una medición del sistema sobre la base $B = \{|1\rangle, \ldots, |N\rangle\}$. El número de iteraciones necesarias para observar una coincidencia con probabilidad no despreciable es, como se espera, $O(\sqrt{N})$. Se puede resumir el algoritmo en el siguiente pseudocódigo.

Entrada: w, $p \in \Sigma^*$
Salida: $m \in [1, N-M+1]$
Variables cuánticas: ψ
Variables clásicas: $r, i, j \in \mathbb{N}$

1. Escoger $r \in [1, \lfloor \sqrt{N-M+1} \rfloor]$ uniformemente,
2. hacer $|\psi_0\rangle = \frac{1}{\sqrt{N-M+1}} \sum_{k=1}^{N-M+1} |k, k+1, \ldots, k+M-1\rangle$;
3. for $i = 1$ to r:
   a) Escoger $j \in [1, M]$ uniformemente.
   b) hacer $|\psi\rangle = I^{\otimes j-1} \otimes Q_{p_j}^{w} \otimes I^{\otimes M-j} |\psi\rangle$ (Cambio de fase);
   c) hacer $|\psi\rangle = (D \otimes I^{\otimes M-1}) |\psi\rangle$ (difusión de Grover)
4. m $\Leftarrow$ medición ψ sobre la base B

Al suponer un oráculo para calcular $Q_{p_j}^{w}$ para todo p_j en el patrón, la complejidad *query* del algoritmo cuántico para *pattern matching* es $O\sqrt{N}$, sin dependencia sobre M, además del costo de preparar el estado inicial de la Ecuación 37.

3.3.3. Complejidad del algoritmo

Basado en un enfoque de *compilar una vez, correr múltiples veces*, el algoritmo de Mateus sólo necesita realizar la preparación de los sub-patrones en superposición una vez, permitiendo una cantidad arbitraria de búsquedas distintas sobre la misma cadena; el algoritmo ofrece una *query complexity* de $O(\sqrt{N})$ en el caso más relevante donde el tamaño del patrón M es mucho más pequeño que el tamaño N de la cadena w. Sólo el costo de la preparación del estado inicial muestra una dependencia sobre M.

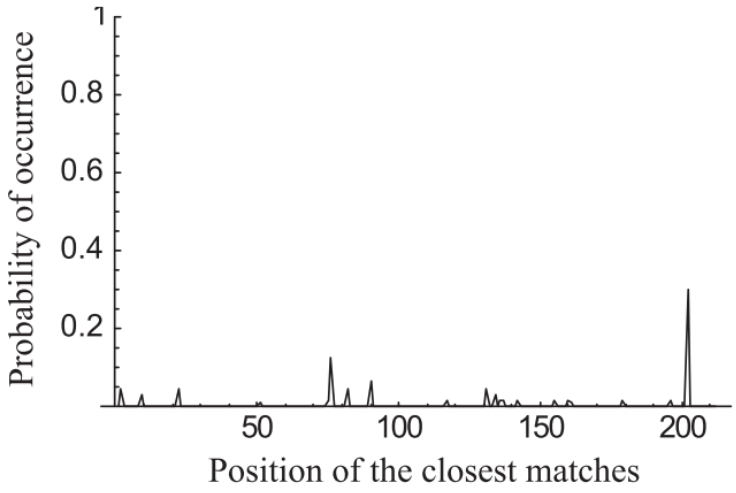

Figura 14. Simulación del algoritmo para una cadena aleatoria de tamaño $N = 212$ y un patrón particular de tamaño $M = 10$ que ocurre hacia el final de la cadena.

La implementación de esta preparación se discute a detalle en Mateus y Omar (2005).

Como cualquier función Booleana de n bits se puede implementar usando $O(n^2 2^n)$ compuertas C-NOT y $O(2^n)$ compuertas Pauli−X, esto significa que un circuito query cuántico para inspeccionar una lista de N elementos se puede construir usando $O(N \log^2(N)) = \tilde{O}(N)$ compuertas. La complejidad de circuito para este algoritmo (excluyendo el estado inicial de la Ecuación 37) es $O(N^{3/2} \log^2(N) \log(M))$ contra $O(MN^2)$ para un circuito clásico (ya que producir un circuito que lea una base de datos arbitraria de tamaño N requiere $O(N)$ compuertas clásicas Booleanas). La complejidad para la construcción del estado inicial de la Ecuación 37 es $O(M \log^3(N - M))$.

Se concluye que el algoritmo tiene un tiempo de compilación de $O(N \log^2(N) \times |\Sigma|)$; si se considera el *tiempo de ejecucion* como la compilación más la iteración de Grover se obtiene una complejidad de $O(M \log(N)^3 + N^{3/2} \log(N)^2 \log(M))$. Este tiempo de ejecución se debe tomar con mucha cautela, ¿Significa que en realidad la solución clásica es más rápida? No, sigue habiendo una mejora cuadrática, en el artículo de Mateus se da la receta completa de una posible construcción del algoritmo a un nivel fundamental; esto es, a nivel compuertas cuánticas. Pero así como en la solución clásica de $O(MN)$ no se toma en cuenta la construcción de la computadora sobre la que corre el algoritmo, ni el tiempo que se toma leer la base de datos desde la memoria, se puede

hacer exactamente lo mismo en la versión cuántica; eliminar esos pasos en los que aún se discute su optimalidad, implementaciones físicas, codificaciones, etc. Este análisis sirve para ejemplificar la utilidad de la *query complexity* para comparar algoritmos cuánticos y clásicos. Sencillamente se considera la complejidad como $O(\sqrt{N})$ respecto a llamadas a oráculos, a la espera de encontrar soluciones a todas las cuestiones que giran alrededor de la computación cuántica y pensar en maneras de cuantificar con precisión los pasos dentro de un algoritmo cuántico.

Capítulo 4. Algoritmo cuántico para el problema de jumbled matching

En Mateus y Omar (2005) los autores proponen un algoritmo para resolver la búsqueda de patrones más cercanos, como en la Definición 12. Se usará la misma preparación del algoritmo de la sección anterior. La entrada del sistema es el patrón de consulta (o el vector de Parikh del patrón) y la cadena de texto, la salida son todos los lugares en el texto donde tenemos un *jumbled match* con el patrón de entrada. Se codifica la cadena en un espacio de Hilbert H, con base $N = \{|1\rangle, |2\rangle, \ldots, |N\rangle\}$, pero antes de que sea posible aplicar el iterador de Grover, es necesario transformar la entrada en una forma tal que cada subcadena en la superposición de la Ecuación 40:

$$|\psi_0\rangle = \frac{1}{\sqrt{N-M+1}} \sum_{k=1}^{N-M+1} |k, k+1, \ldots, k+M-1\rangle \tag{40}$$

es un elemento de un nuevo alfabeto Σ', donde Σ' es el alfabeto de los vectores de Parikh posibles. Claramente, dos subcadenas en superposición que hacen *jumbled match* se convierten en el mismo vector de Parik; por tanto, el nuevo alfabeto tiene un tamaño $|\Sigma'| \leq N - M + 1$.

Cada estado en la superposición de la Ecuación 40 que representa la subcadena en la ventana deslizante se traduce a un vector de Parikh por un operador U_p como en la ecuación 41:

$$U_p |k, k+1, \ldots, k+M-1\rangle = |P_k\rangle \tag{41}$$

donde P_k es el vector de Parikh correspondiente.

4.1. Procesamiento de la preparación de Mateus

Se plantearán dos posibles soluciones al problema de *jumbled matching* en el paradigma de la computación cuántica; la primera de estas soluciones es construir la versión reversible de un operador de transformación en forma de grafo. La segunda

variante es una aplicación de un operador basado en primalidad. A continuación se presentan ambas aproximaciones.

4.1.1. Primera solución

La primera pieza del operador de transformación es un grafo completo bipartito $U_p = (\Sigma, S)$ con S un marcador para subcadenas de tamaño $|p|$ (ver la figura 15). El operador de transformación depende del patrón de consulta p; como el patrón es dado en el problema, el operador correspondiente se prepara junto con el texto. En la Figura 15 se puede ver un ejemplo de este operador como un grafo bipartito, donde la subcadena en superposición está a la izquierda y el alfabeto original a la derecha; cada arista representa un *AND* lógico, después de aplicar las operaciones *AND* las aristas sobrevivientes cuentan las multiplicidades de los elementos de Σ en la subcadena.

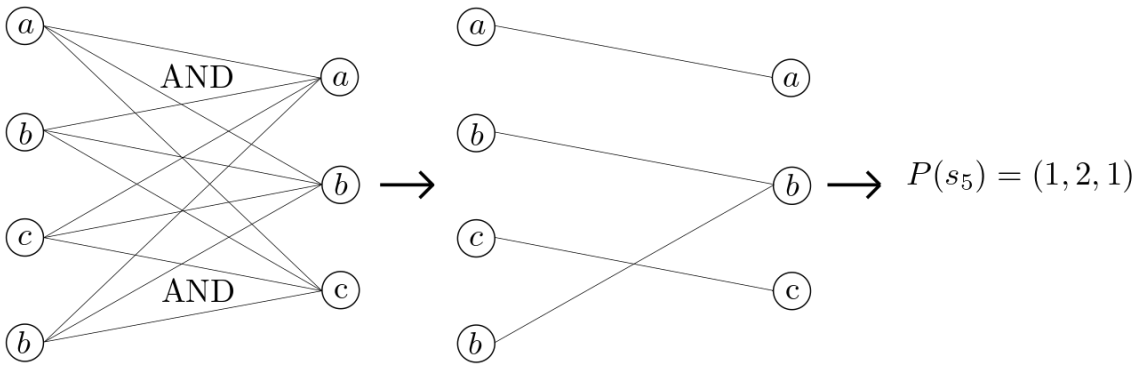

Figura 15. Oráculo que calcula el vector de Parikh de la subcadena en superposición.

El esquema de construcción para el operador de transformación del grafo se puede observar en la Figura 16. Por un lado, ingresa el sub-patrón desde la preparación de la Ecuación 40, y en un registro auxiliar se tendrían inicializados los elementos del alfabeto original. Las compuertas vacías representan las versiones reversibles de las compuertas AND necesarias para generar los vectores de Parikh. En el ejemplo de la Figura 16, el sub-patrón *abca* debe generar el vector de Parikh $(2, 1, 1)$ pues está construído por 2 *a*'s, 1 *b* y 1 *c*. La salida del primer bloque, después de la aplicación de las compuertas AND es el elemento *a* del alfabeto, la salida del segundo bloque es el elemento *b*, la salida del tercer bloque es el elemento *c* y la salida del cuarto bloque

es; de nuevo, el elemento a. Por tanto se genera la salida $abca$, que ahora se tratará como el vector de Parikh $(2, 1, 1)$. Aunque esto pueda parecer una simple reescritura, es clave notar que cualquier permutación del subpatrón $abca$ va a generar la misma salida. Con esta construcción se obtiene un circuito que traducirá los sub-patrones tal como en la Ecuación 41.

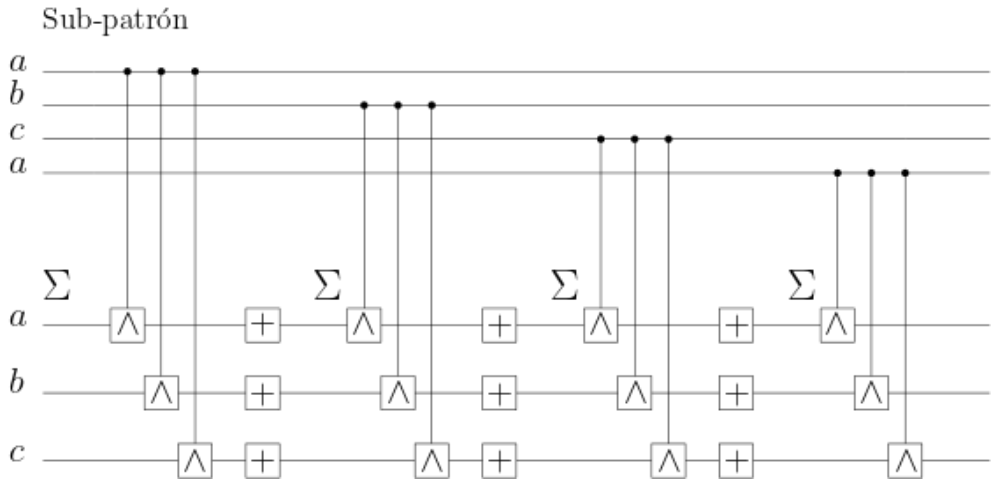

Figura 16. Construcción de alto nivel del operador de transformación presentado en la figura 15, las compuertas ∧ son *AND* reversibles y las compuertas + son sumadores reversibles.

4.1.2. Segunda solución

En la segunda solución se considera una codificación distinta para la cadena w y el patrón p, de modo que a cada elemento del alfabeto original Σ le corresponderá un único número primo.

Dado un conjunto ordenado de números primos $\{p_1, p_2, \ldots, p_{|\Sigma|}\}$, cada elemento $\sigma \in \Sigma$ se reescribe como

$$\begin{aligned} \sigma_1 &\Rightarrow p_1 \\ \sigma_1 &\Rightarrow p_1 \\ &\vdots \\ \sigma_{|\Sigma|} &\Rightarrow p_{|\Sigma|} \end{aligned} \tag{42}$$

Así, esta codificación hace que w y p sean cadenas conformadas por números primos; no hay ningún costo extra para el algoritmo cuántico dado que la codificación se da junto con el problema. Con esto, hasta antes de la preparación de Mateus (40) todo queda igual, se codificará la posición i en un vector unitario $|i\rangle$ de un espacio de Hilbert **H** de dimensión N, donde el conjunto $B = \{|1\rangle, |2\rangle, \ldots, |N\rangle\}$ constituye una base ortonormal de **H**.

Superposición de vectores de Parikh

Siguiendo la receta de la preparación inicial de Mateus de la Ecuación 40, ahora se tiene una superposición donde cada *ventana* o sub-patrón en superposición se constituye de elementos del alfabeto Σ de números primos. El paso siguiente es convertir cada sub-patrón en vectores de Parikh, además, hay que notar que cada sub-patrón ahora es una secuencia de números.

Considérese el siguiente teorema:

Teorema 4.1 *Sean dos cadenas $\alpha \in \Sigma^*$ y $\beta \in \Sigma^*$, con $\Sigma = \{2, 3, 5, \ldots, P_{|\Sigma|}\}$ un alfabeto conformado por el conjunto de los primeros $|\Sigma|$ números primos.*

Se cumple que los vectores de Parikh de ambas cadenas son iguales si, y sólo si el producto de los caracteres que conforman las cadenas son iguales, es decir:

$$P(\alpha) = P(\beta) \iff \prod_i \alpha_i = \prod_i \beta_i.$$

Se define un operador que toma como entrada un sub-patrón de la superposición de Mateus y calcula su producto:

Definición 13 *Sea el operador de multiplicación Q_Π aquel que al aplicarse sobre el sub-patrón $|\alpha_k, \alpha_{k+1}, \ldots, \alpha_{k+M-1}\rangle$ genera el producto de sus caracteres:*

$$Q_\Pi |\alpha_k, \alpha_{k+1}, \ldots, \alpha_{k+M-1}\rangle = |\alpha_k \cdot (\alpha_{k+1}) \cdots (\alpha_{k+M-1})\rangle$$

Este operador se puede aplicar en un sólo paso sobre la superposición de Mateus. Obteniendo a la salida una superposición de vectores de Parikh

$$|\Psi_1\rangle = \frac{1}{\sqrt{N-M+1}} \sum_{k=1}^{N-M+1} |P_k\rangle \tag{43}$$

Se puede ver que los vectores de Parikh en esta superposición tienen dimensión 1; es decir, simplemente números. Esto debido a la codificación que se definió previamente.

Una forma directa de implementar este operador es crear un circuito clásico para realizar multiplicación a apartir de compuertas universales; por ejemplo, usando compuertas NAND, y reemplazándolas por compuertas Toffoli. Existen multiples artículos relacionados con este problema de implementabilidad, como en Banerjee y Pathak (2009) donde se dan algunas recetas de multiplicadores y sumadores reversibles que se pueden construir con las compuertas de la librería *NCT*; es decir, compuertas NOT, CNOT y CCNOT o Toffoli.

Aplicación del iterador de Grover

Ahora que se tienen todos los vectores de Parikh en superposición, lo único que resta es aplicar el iterador de Grover para multiples ocurrencias (Boyer *et al.* (1996). Para este fin, se debe modificar la función query de la Ecuación 38. Esta modificación se define por la Ecuación 44:

$$f_\sigma(i) = \begin{cases} 1 & \text{si el } i-\text{ésimo vector de Parikh de } w \text{ es } \sigma \\ 0 & \text{de otra forma.} \end{cases} \tag{44}$$

El oráculo definido por esta función query ahora cambia de fase a cada uno de los

estados cuyo vector de Parikh coincida con el vector de Parikh de la consulta; Y, como ya se discutió en secciones anteriores, será suficiente aplicar el iterador de Grover $O(\sqrt{N})$ veces para observar, con alta probabilidad, un *jumbled match* del patrón p. La medición del registro se realiza, sobre la base B de índices, obteniendo las posiciones donde suceden tales coincidencias. Se puede ver el circuito final del algoritmo para jumbled matching en la Figura 17.

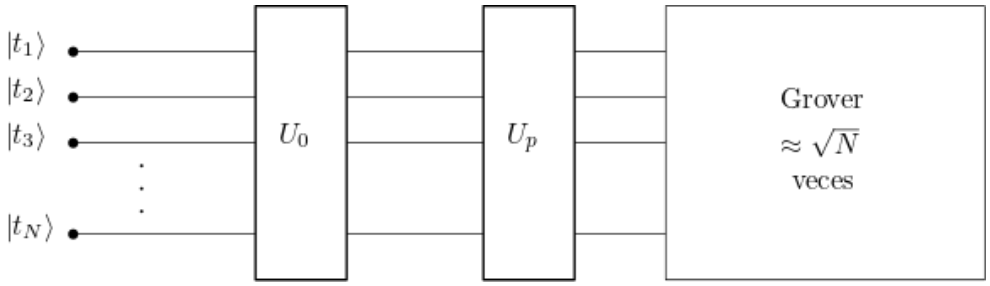

Figura 17. Circuito de alto nivel para el algoritmo de jumbled matching, donde el primer operador genera la superposición de Mateus, la segunda realiza la conversión a vectores de Parikh (con alguna de las dos formas propuestas) y el tercer oráculo siendo el iterador de Grover.

4.2. Complejidad de las soluciones propuestas

La complejidad del algoritmo es idéntica a la del algoritmo de Mateus. El único añadido al algoritmo de *pattern matching* presentado anteriormente es la llamada al operador que genera la multiplicación de los elementos del sub patrón (o la obtención del vector de Parikh con el grafo para la primera solución), en la que usando el paralelismo cuántico opera en una sola llamada, simultáneamente sobre cada uno de los sub-patrones en superposición. Después de la traducción a vectores de Parikh la aplicación del iterador de Grover se realiza $O(\sqrt{N})$ veces; por tanto, la *query complexity* de nuestro algoritmo es $O(\sqrt{N})$. Igual que en el algoritmo de Mateus usamos el enfoque de *compilar una vez, correr múltiples veces*, donde la preparación inicial se hace sólo una vez, ofreciendo la posibilidad de realizar muchas búsquedas distintas. Se justifica la complejidad del algoritmo bajo los mismos criterios con los que se llega a la complejidad $O(\sqrt{N})$ del algoritmo de Mateus, eliminando las sutilezas de la propia máquina (el circuito) que implementaría este algoritmo.

Capítulo 5. Conclusiones y trabajo futuro

Se hace énfasis en que el diseño de algoritmos cuánticos es un arte, no existe una metodología bien definida para su construcción y aún se está en el punto donde se debe construir una computadora de propósito específico para cada problema que se desea resolver.

No se ha estudiado la bibliografía básica del área, y con ello simplemente tomado primitivas para aplicarlas de una forma mecánica en la solución del problema de *jumbled matching*. La exploración de los algoritmos cuánticos y la teoría de la computación cuántica en general que se llevó a cabo en este trabajo, ha resultado en el descubrimiento de infinidad de asuntos por resolver en la teoría de la computación. La necesidad de tener múltiples artículos a la mano con cada párrafo leído, comprender con el mayor detalle posible las definiciones, teoremas y corolarios ha contribuido a un entendimiento del cómo los efectos cuánticos se pueden usar; no sólo para la creación de algoritmos, sino para ver a los clásicos problemas de complejidad y computación teórica desde un enfoque más amplio. Uno donde cabe preguntarse si la mecánica cuántica dará herramientas para escapar de la cada vez más cercana disrupción de la Ley de Moore; o, si acaso se está a las puertas de una era en la que los problemas NP-Completos dejen dormir con toda tranquilidad. Gracias a la computación cuántica se encontraron tópicos tan interesantes como la relativización de las clases de complejidad, reducciones y pruebas de cotas inferiores; el modelo de la *query complexity*, el área tan complicada y con cientos de obstáculos por resolver de la implementabilidad fisica de los algoritmos. Infinidad de tópicos que no se pueden abarcar o siquiera escribir en el marco teórico de esta tesis de maestría, pero que han aportado al único fin de resolver el problema planteado al inicio de este emprendimiento.

En este trabajo se logró diseñar un algoritmo cuántico para el problema de *jumbled pattern matching*, que ofrece una aceleración cuadrática respecto a la solución clásica. En una primera variante del algoritmo, se propuso un grafo que calcula los vectores de Parikh directamente desde la codificación de los sub-patrones en el alfabeto original de las cadenas, cuya aplicación se considera una llamada a oráculo. En la segunda

variante se ideó una codificación distinta para las cadenas originales, donde ahora son codificadas en un alfabeto nuevo que es conformado por números primos. Mediante esta técnica se logró que un operador de multiplicación se aplique en un sólo paso, ahorrando la construcción explícita del operador generador de vectores de Parikh de la primera variante.

A partir de esta tesis se hallaron posibles rutas por explorar en trabajo futuro, de forma inmediata se podría tratar de resolver el mismo problema desde el punto de vista de *indexamiento cuántico*. Dado que para *jumbled matching* en línea existe una aceleración cuadrática en cómputo cuántico, si se supone la existencia de un modelo de direccionamiento cuántico análogo a la RAM clásica, sería posible preguntarse lo siguiente:

- ¿Es posible acelerar la solución clásica de *jumbled indexing* en una computadora cuántica?
- ¿Existe el análogo cuántico del teorema para la dureza condicional presentado en Amir *et al.* (2014)?

Como un paso minúsculo hacia la solución de estas cuestiones, se presenta de forma breve cuál podría ser el proceso para realizar llamadas a superposiciones contenidas en una posible QRAM. Estas superposiciones podrían actuar como los estados que codifican ciertos sub-patrones que hacen *jumbled match* con un patrón de consulta dado.

Codificación vía QRAM

Aunque aún se encuentran en desarrollo la técnica y teoría de la generación de registros cuánticos con la información clásica requerida por los algoritmos, valdría la pena mencionar una forma posible en la que podría realizarse la codificación arriba mencionada. Para el algoritmo de Grover, por ejemplo, se requeriría una superposición de la forma

$$|\Psi\rangle = \frac{1}{\sqrt{N}} \sum_x |x, w_x\rangle \in \mathcal{H}_{\text{index}} \otimes \mathcal{H}_{\text{entry}},$$

donde cada elemento de la superposición vive en un espacio conformado por el es-

pacio de Hilbert de los índices y el espacio de Hilbert de las entradas, en esta superposición x es el índice del elemento o *palabra* de entrada $|w_x\rangle$. La generación de un estado como estos es altamente no trivial, pero se puede suponer la existencia de un operador que usa n qubits para direccionar cualquier superposición cuántica de N celdas a partir de una RAM. A este sistema se le llama *QRAM* (para una discusión más amplia de la QRAM ver Giovannetti *et al.* (2008)).

El operador Q implementaría la evolución de la Equación 45.

$$Q(|x\rangle_{\text{address}}|0\rangle_{value}) \Rightarrow |x\rangle_{address}|w_x \oplus 0\rangle_{value} = |x\rangle_{address}|w_x\rangle_{value} \tag{45}$$

Explícitamente, si tenemos dos registros inicializados en el estado $|0\rangle$, se aplicaría una compuerta de Hadamard en el primer registro, luego el operador de carga, para finalmente obtener la preparación de la Ecuación 9:

$$H^{\oplus n}_{address}|0\rangle|0\rangle = \frac{1}{2^{n/2}}\sum_{x=0}^{2^n-1}|x\rangle|0\rangle \xrightarrow{Q} \frac{1}{2^{n/2}}\sum_{x=0}^{2^n-1}|x\rangle|w_x\rangle\,. \tag{46}$$

Entonces, al aplicar un operador query, la función tratará al índice como un apuntador hacia el contenido del registro. Así, se pueden aplicar operadores query sobre el registro de índices suponiendo que en todo momento realmente se está tratando con el contenido de la base de datos, en este caso, los caracteres de la cadena T.

Otra posible, y muy interesante línea de investigación sería traducir el problema de *jumbled matching* a un problema de optimización. Por ejemplo, usando la distancia L_1 entre los vectores de Parikh de dos ventanas de texto como función objetivo, con el propósito de obtener el mínimo; en este caso, cero (hacer jumbled match). Una vez definido el problema de optimización explorar el uso de computadoras de propósito específico basadas en la computación cuántica adiabática. Este modelo de cómputo utiliza un conjunto de estados cuánticos y un Hamiltoniano dependiente del tiempo $H(t)$ tal que el estado base del Hamiltoniano $H(t_f)$ en un tiempo t_f codifica la solución del problema en cuestión; por el *teorema adiabático*, si se tiene un sistema cuántico en el estado $H(t_0)$ tal que evoluciona a $H(t_f)$, permanecerá en el estado base de $H(t)$ para todo t siempre que la evolución se haga lo suficientemente lenta. Al final de este proceso una medición sobre el sistema generará la solución a nuestro problema con

alta probabilidad. Una vez se encuentre la versión de optimización adecuada, se podría utilizar un sistema como la D-Wave 2000Q e intentar la implementación de esta forma de *jumble matching*. Para más información sobre el cómputo cuántico adiabático ver Farhi *et al.* (2000), Boixo *et al.* (2014).

Literatura citada

Aaronson, S. (2013). *Quantum Computing since Democritus*. Cambridge University Press. USA.

Ambainis, A. (2017). Understanding quantum algorithms via query complexity. *Computing Research Repository*. Recuperado el 10/02/2021, de http://arxiv.org/abs/1712.06349.

Amir, A., Chan, T., Lewenstein, M., y Lewenstein, N. (2014). On hardness of jumbled indexing. En: *ICALP*.

Banerjee, A. y Pathak, A. (2009). An analysis of reversible multiplier circuits. *arXiv: Quantum Physics*.

Bennett, C. H., Bernstein, E., Brassard, G., y Vazirani, U. (1997). Strengths and weaknesses of quantum computing. *SIAM J. Comput.*, **26**: 1510–1523.

Bernstein, E. y Vazirani, U. (1997). Quantum complexity theory. *SIAM J. Comput.*, **26**: 1411–1473.

Boixo, S., Ronnow, T. F., Isakov, S., Wang, Z., Wecker, D., Lidar, D., Martinis, J., y Troyer, M. (2014). Evidence for quantum annealing with more than one hundred qubits. *Nature Physics*, **10**: 218–224.

Boyer, M., Brassard, G., Hoeyer, P., y Tapp, A. (1996). Tight bounds on quantum searching. *Protein Science*, **46**: 493–505.

Brassard, G., Høyer, P., Mosca, M., Montreal, A., Aarhus, B. U. O., y of Waterloo, C. U. (2000). Quantum amplitude amplification and estimation. *arXiv: Quantum Physics*. Recuperado el 20/10/2020, de https://arxiv.org/abs/quant-ph/0005055.

Burcsi, P., Cicalese, F., Fici, G., y Lipták, Z. (2010). On table arrangements, scrabble freaks, and jumbled pattern matching. En: *FUN*.

Church, A. (1936). An unsolvable problem of elementary number theory. *American Journal of Mathematics*, **58**: 345.

Cicalese, F., Fici, G., y Lipták, Z. (2009). Searching for jumbled patterns in strings. En: *Proceedings of the Prague Stringology Conference 2009*. pp. 105–117.

Cook, S. A. y Reckhow, R. A. (1972). Time-bounded random access machines. En: *Proceedings of the Fourth Annual ACM Symposium on Theory of Computing*, New York, NY, USA. Association for Computing Machinery, STOC '72, p. 73–80.

Cormen, T. H. (2009). *Introduction to algorithms*. MIT press.

Deutsch, D. (1985). Quantum theory, the church–turing principle and the universal quantum computer. *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences*, **400**: 117 – 97.

Farhi, E., Goldstone, J., Gutmann, S., y Sipser, M. (2000). Quantum computation by adiabatic evolution. *arXiv: Quantum Physics*. Recuperado el 03/05/2021, de https://arxiv.org/abs/quant-ph/0001106.

Feynman, R. (1982). Simulating physics with computers. *International Journal of Theoretical Physics*, **21**: 467–488.

Giaquinta, E. y Grabowski, S. (2013). New algorithms for binary jumbled pattern matching. *Inf. Process. Lett.*, **113**: 538–542.

Giovannetti, V., Lloyd, S., y Maccone, L. (2008). Quantum random access memory. *Physical review letters*, **100 16**: 160501.

Gödel, K. (1931). Über formal unentscheidbare sätze der principia mathematica und verwandter systeme i. *Monatshefte für Mathematik und Physik*, **38**: 173–198.

Grover, L. K. (1996). A fast quantum mechanical algorithm for database search. En: *STOC '96*.

Hariharan, R. y Vinay, V. (2003). String matching in õ(sqrt(n)+sqrt(m)) quantum time. *J. Discrete Algorithms*, **1**: 103–110.

Hollenberg, L. (2000). Fast quantum search algorithms in protein sequence comparisons: quantum bioinformatics. *Physical review. E, Statistical physics, plasmas, fluids, and related interdisciplinary topics*, **62 5 Pt B**: 7532–5.

Jørgensen, A. y Pettie, S. (2014). Threesomes, degenerates, and love triangles. *2014 IEEE 55th Annual Symposium on Foundations of Computer Science*, pp. 621–630.

Kak, S. (2017). On quantum decision trees. *ArXiv*, **abs/1703.03693**. Recuperado el 05/11/2020, de `https://arxiv.org/abs/1703.03693`.

Mateus, P. y Omar, Y. (2005). Quantum pattern matching. *arXiv: Quantum Physics*. Recuperado el 20/09/2020, de `https://arxiv.org/abs/quant-ph/0508237`.

Moosa, T. M. y Rahman, M. S. (2010). Indexing permutations for binary strings. *Inf. Process. Lett.*, **110**: 795–798.

Moosa, T. M. y Rahman, M. S. (2012). Sub-quadratic time and linear space data structures for permutation matching in binary strings. *J. Discrete Algorithms*, **10**: 5–9.

Nielsen, M. y Chuang, I. (2010). *Quantum Computation and Quantum Information (10th Anniversary edition)*. Cambridge University Press.

Patrascu, M. (2010). Towards polynomial lower bounds for dynamic problems. En: *STOC '10*.

Preskill, J. (2018). Quantum computing in the nisq era and beyond. *arXiv: Quantum Physics*.

Rahimi-Keshari, S., Ralph, T. C., y Caves, C. M. (2016). Sufficient conditions for efficient classical simulation of quantum optics. *Phys. Rev. X*, **6**: 021039.

Shor, P. (1999). Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J. Comput.*, **26**: 1484–1509.

Soni, K. y Rasool, A. (2020). Pattern matching: A quantum oriented approach. *Procedia Computer Science*, **167**: 1991–2002.

Turing, A. (1937). On computable numbers, with an application to the entscheidungsproblem. *Proceedings of The London Mathematical Society*, **41**: 230–265.

Williams, C. P. y Clearwater, S. (1997). *Explorations in quantum computing*. Springer.

Williams, V. V. (2015). Hardness of easy problems: Basing hardness on popular conjectures such as the strong exponential time hypothesis (invited talk). En: *IPEC*.

Yanofsky, N. S. y Mannucci, M. A. (2008). *Quantum Computing for Computer Scientists*. Cambridge University Press.

Yao, A. (1993). Quantum circuit complexity. En: *FOCS*.

Zhang, A., Wang, X., y mei Zhao, S. (2020). The multiplier based on quantum fourier transform. *CCF Trans. High Perform. Comput.*, **2**: 221–227.

Anexo: Complejidad Computacional

La complejidad computacional mide el número de pasos (que es proporcional al tiempo) o la cantidad de memoria requerida (proporcional al espacio) necesaria para resolver un problema. La complejidad se ocupa de la eficiencia con la que las computadoras pueden resolver tareas computacionales, el hecho de que una computadora pueda resolver cierto tipo de problema, en principio, no garantiza que puede resolverlo en la práctica. Para medir la eficiencia se emplea la tasa de crecimiento en el tiempo o la memoria requerida para resolver un problema respecto al tamaño de su entrada. Por ejemplo, si un algoritmo es usado para resolver un problema de búsqueda en una base datos de N elementos, N sería el tamaño de la entrada. La distinción tradicional de complejidad computacional entre problemas *tratables* y *no tratables* dependen de si la escala asintótica del algoritmo crece polinomialmente; es decir, $O(n^k)$, o exponencialmente $O(k^n)$ con el tamaño de entrada n.

.1. Notación Big-O

La teoría de la complejidad involucra precisar cómo se comportan los algoritmos en el límite asintótico. Esto se hace comparando la tasa de crecimiento del algoritmo con una función matemática simple en el límite, donde el tamaño del problema computacional tiende al infinito.

Consideremos tres funciones $f(x) = \sqrt{\frac{x}{2}}, g(x) = \frac{3}{x}\sin x + \log x$, y $h(x) = \log \frac{3x}{4}$. Se pueden ver sus gráficas en la Figura 18. Para valores pequeños de $x, g(x)$ puede ser más grande o menor que f(x), y probablemente más grande o menos que $h(x)$. Sin embargo, asintóticamente, $g(x)$ es acotada por arriba por $f(x)$ y por tanto $g(x) = O(f(x))$. Similarmente, asintóticamente, g(x) es acotada por abajo por $h(x)$, así $g(x) = \Omega(h(x))$. Esta notación se usa para caracterizar el comportamiento asintótico de algoritmos. En la Tabla 1 vemos un resumen de las relaciones de escala más comunes en la notación *Big-O*.

Otra notación muy común es aquella que esconde los factores logarítmicos, llamada *O suave* o *soft-O*; es decir, una función es $f(n) = \tilde{O}(g(n))$ si $f(n) = O(g(n)\log^k(g(n)))$.

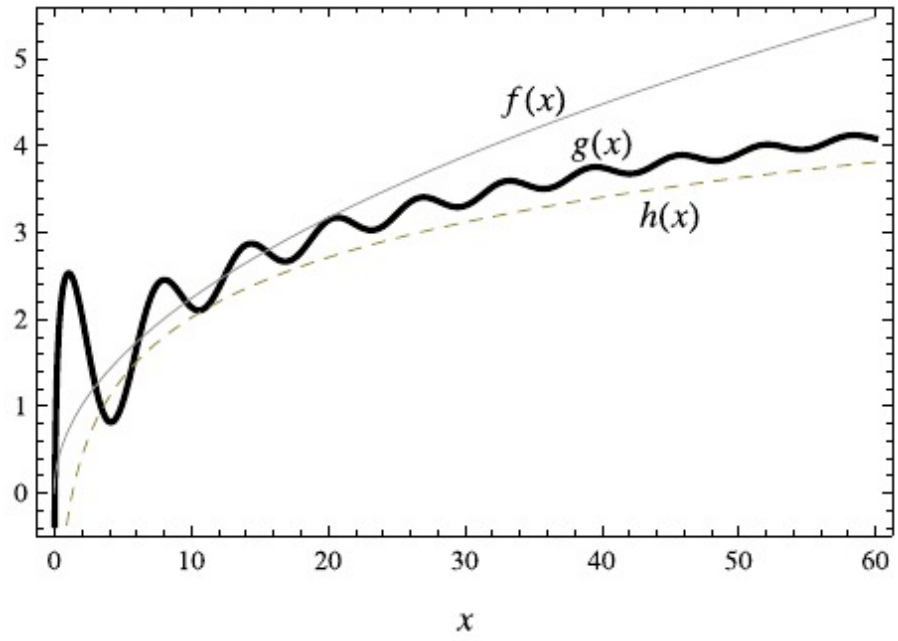

Figura 18. Crecimiento de tres funciones distintas conforme crece el tamaño de la entrada.

Tabla 1. Notación usada para caracterizar el comportamiento de escala asintótica de algoritmos.

Notación	Significado	Definición Formal
$f(x) = O(g(x))$	$f(x)$ es acotada por arriba por $g(x)$ asintóticamente	Como $x \to \infty$, $\exists k$, t.q $\lvert f(x)\rvert \leq kg(x)$
$f(x) = o(g(x))$	$f(x)$ es dominada por $g(x)$ asintóticamente	Como $x \to \infty$, $\forall k$, $\lvert f(x)\rvert \leq kg(x)$
$f(x) = \Omega(g(x))$	$f(x)$ es acotada por abajo por $g(x)$ asintóticamente	Como $x \to \infty$, $\exists k$, t.q $\lvert f(x)\rvert \geq kg(x)$
$f(x) = \omega(g(x))$	$f(x)$ domina $g(x)$ asintóticamente	Como $x \to \infty$, $\forall k$, $\lvert f(x)\rvert \geq kg(x)$
$f(x) = \Theta(g(x))$	$f(x)$ es acotada por arriba y por abajo por $g(x)$ asintóticamente	Como $x \to \infty$, $\exists k_1, k_2$, t.q $k_1 g(x) \leq \lvert f(x)\rvert \leq k_2 g(x)$
$f(x) \sim g(x)$	$f(x)$ iguala $g(x)$ asintóticamente	Como $n \to \infty, \forall k$, $\lvert f(x)/g(x) - 1\rvert \leq k$