Universidad Nacional Autónoma de México

Facultad de Ciencias

Programación Automática de Computadoras Cuánticas por medio de Programación Genética Lineal

T E S I S

que para obtener el grado de:
Físico

presenta:
Mario Oscar Franco Méndez

director de tesis:
Dra. Katya Rodríguez Vázquez

Ciudad de México Marzo, 2019

Programación Automática de Computadoras Cuánticas por medio de Programación Genética Lineal

por

Mario Oscar Franco Méndez

Tesis presentada para obtener el grado de

Físico

en la

Facultad de Ciencias

Universidad Nacional Autónoma de México

Ciudad de México. Marzo, 2019

Agradecimientos

A la Dra. Katya Rodríguez Vázquez por la dirección de este trabajo, así como por el conocimiento compartido y su amistad.

A los Drs. Ramón López Peña, Pablo Barberis Blostein, Carlos Francisco Pineda Zorrilla y Gustavo De la Cruz Martínez por sus observaciones y correcciones prestadas para el presente trabajo.

A la Universidad Nacional Autónoma de México y, en particular, a la Facultad de Ciencias por brindarme la oportunidad de realizar mis estudios profesionales.

Índice general

Resumen

Debido a la gran promesa que el cómputo cuántico ofrece y gracias a los avances que en la actualidad se han dado en dirección a su realización física, es cada vez más importante el desarrollo de algoritmos cuánticos, en especial, algoritmos cuánticos eficientes. Con esta idea en mente, en el presente trabajo se diseño un sistema capaz de diseñar algoritmos cuánticos de manera automática. El sistema consiste de un algoritmo de programación genética lineal y una simulación de una computadora cuántica.

El sistema fue puesto a prueba con seis problemas diferentes. En primer lugar, se utilizó el sistema para crear un algoritmo capaz de identificar entre oráculos balanceados y oráculos constantes, esto es, un algoritmo similar al de Deutsch Josza; en segundo lugar, se buscó un algoritmo capaz de encontrar un elemento de una base de datos desordena, de manera similar al algoritmo de búsqueda de Grover; los últimos cuatro problemas consistieron en buscar algoritmos para evaluar funciones booleanas compuestas por operadores AND y OR. En estos últimos problemas se realizó se estudió la utilidad de mediciones parciales de los qubits a la mitad del algoritmo para ramificar el comportamiento del algoritmo. En todos los problemas anteriormente mencionados se realizaron 20 corridas de las cuales se seleccionó al mejor algoritmo y se simplificó manualmente.

En general, fue posible obtener algoritmos funcionales para todos los problemas aquí planteados. Para los problemas de Deutsch Josza y de Grover se encontraron soluciones perfectas en cada una de las veinte corridas. Para la evaluación de las cuatro funciones booleanas, con

y sin mediciones parciales a mitad del circuito, se encontraron algoritmos con probabilidad de error menor respecto del mejor algoritmo probabilístico clásico posible en cada una de las veinte corridas, aunque no se encontró ninguna mejoría en la calidad de los algoritmos con la utilización mediciones parciales a la mitad del circuito. Se observó que la mayoría de los algoritmos desarrollados por este sistema utilizan una superposición pesada en el qubit objetivo de la función oráculo.

Introducción

2.1. La Máquina de Turing

Desde su postulación por Alan Turing en 1936, las máquinas de Turing han captado la atención de un gran número de científicos, creando diversas nuevas ramas del conocimiento, y han cambiado radicalmente la vida del hombre a tal punto que el estilo de vida contemporáneo sería inimaginable sin las máquinas de Turing o, como llamamos a su versión física, computadoras.

De manera un poco más formal, una máquina de Turing es un dispositivo capaz de procesar algoritmos, de manera similar a como lo haría una persona, compuesto por tres elementos fundamentales: una cinta, 'infinita', compuesta a su vez de diversas casillas que pueden o no contener una letra x de un alfabeto $\{\alpha, \beta, \ldots\}$; una unidad de control con un número finito de estados $\{s_0, \ldots, s_k, H\}$, donde H denota el estado para la cual dicha máquina se detiene; y, por último, un sistema para leer y escribir en una casilla de la cinta, capaz de desplazarse de una casilla a otra. El cómputo realizado por una máquina de Turing está determinado por un programa, es decir, un conjunto finito de instrucciones.

En un dispositivo de tales características podemos distinguir solamente tres tipos de acciones diferentes: la transición del estado de la unidad de control de s a $\hat{s}$; la transición de la letra de una casilla de a a $\hat{a}$; y el desplazamiento del sistema de lectura y escritura d a la izquierda o a la derecha. Por lo que la acción de una instrucción es descrita por un mapeo: $(s, a) \mapsto (\hat{s}, \hat{a}, d)$.

Una máquina de Turing probablilística difiere únicamente, con respecto de la máquina descrita anteriormente o máquina de Turing determinista, en que el mapeo $(s, a) \mapsto (\hat{s}, \hat{a}, d)$ es probabilístico. Una máquina de Turing probabilística es capaz de resolver diversos problemas más rápido que una máquina de Turing determinista; esto no significa que el conjunto de problemas que una máquina de Turing probabilística es capaz de resolver sea más grande que el conjunto que resuelve su contraparte determinista. De hecho, una máquina de Turing probabilística se puede simular con una máquina de Turing determinista.

2.2. La Máquina de Turing Cuántica

La máquina de Turing surgió bajo una concepción clásica de la naturaleza, una en la que los fenómenos cuánticos se pasaron por alto. Por lo que tiempo atrás surgió la pregunta de si era posible reformular la idea de la máquina de Turing para incluir las 'nuevas' posibilidades de la naturaleza. En el pasado siglo, varios físicos importantes, entre ellos: Feynman, Benioff, Bennett, Deutsch, entre muchos más, se plantearon esta pregunta, dando origen a la máquina de Turing cuántica.

Una manera de ver a una máquina de Turing cuántica es como un sistema análogo a una máquina de Turing probabilística pero que en lugar de utilizar probabilidades tradicionales utiliza amplitudes de probabilidad complejas aunada a la restricción de que el cómputo debe ser reversible.

Otra posibilidad, es ver a la máquina de Turing cuántica como un dispositivo que aplica transformaciones unitarias a un espacio complejo de superposiciones de estados. Visto de esta forma, un algoritmo cuántico se puede pensar como una descomposición de una transformación unitaria en transformaciones unitarias locales o transformaciones unitarias intuitivas para un humano.

Vale la pena mencionar que la restricción de que el cómputo sea reversible o, equivalentemente, que las transformaciones sean unitarias no es debido a un capricho humano sino más

bien de la naturaleza, ya que la evolución de un sistema cuántico se comporta en concordancia con éste tipo de transformaciones.

En las últimas décadas se han realizado diversos trabajos demostrando el poder de las máquinas de Turing cuánticas, o computadoras cuánticas, frente a su contraparte clásica, la máquina de Turing probabilística, como los trabajos de Deustch, Grover, Shor, por nombrar unos pocos, por lo que el computo cuántico ha levantado altas expectativas alrededor de todo el mundo como un paradigma capaz de revolucionar la sociedad humana como anteriormente lo hizo la idea original de Alan Turing. Sin embargo, aun resta mucho trabajo por realizarse antes de poder ver los frutos del computo cuántico.

2.3. Programación Genética

El mundo natural es, desde tiempos inmemoriales, una fuente de inspiración para todas las ramas de la actividad humana: arte, ciencia, filosofía, etc. Una rama que ha encontrado esta relación con la naturaleza especialmente fructífera es el cómputo, dando lugar a una gran diversidad de técnicas y algoritmos basados en situaciones observadas en la naturaleza como lo es el algoritmo genético, basado en el proceso de selección natural de las especies.

La programación genética es una familia de algoritmos basados en el algoritmo genético donde una serie de programas o algoritmos son sometidos a un proceso similar al de selección natural. Curiosamente, una de las primeras personas en sugerir la evolución de programas fue Turing, aunque tuvieron que pasar alrededor de treinta años para que R. Forsyth lograra evolucionar un conjunto de gráficas de árboles, o programas, para resolver un problema de clasificación.

Actualmente y gracias al aumento considerable de poder computacional, la programación genética es capaz de dar respuesta a problemas sumamente complejos, obtener algoritmos que en muchos casos superan al obtenido por una persona e incluso dar respuesta a problemas a los que ninguna persona había podido darle respuesta.

2.4. Motivaciones

Aunque el cómputo cuántico presenta grandes promesas para el futuro, está acompañado de muchos problemas de una gran dificultad, tanto en la parte física, es decir, su implementación en el mundo natural; como en la manera de utilizarlas, ésto es, la manipulación de la información y el diseño de algoritmos eficientes. El presente trabajo intenta abordar este último punto: el diseño de algoritmos cuánticos eficientes.

Por un lado, el diseño de algoritmos eficientes es un problema de gran complejidad que, en muchos casos, se asemeja más a un trabajo creativo, como la creación de un lienzo o una pieza musical, aunado a una serie de accidentes acertados que a un trabajo matemático riguroso.

Por el otro, los humanos somos entes inmersos en un mundo que, para todo fin práctico, se comporta de manera clásica, es decir, seres con una concepción clásica del universo, por lo que la creación de algoritmos cuánticos presenta un problema extra: ¿cómo impedir que nuestra concepción clásica del universo intente imponerse en un sistema que se comporta de todas formas salvo clásicamente?.

Debido a estas dos problemáticas, en el presente trabajo se plantea a la programación genética como una herramienta que tal vez pueda ayudar a sortear estas dificultades, aunque sin llegar a ser un remplazo puesto que aún no se ha avanzado tanto en esta técnica como para que pueda serlo o incluso si quiera si tiene la capacidad de serlo, sino mas bien como un auxiliar al momento de explorar todas las posibilidades para el diseño de algoritmos cuánticos ya que es posible que podamos aprender como diseñar nuevos algoritmos observando lo que se desarrolla en una naturaleza de juguete.

Computación Cuántica

El cómputo cuántico es un paradigma de computación basado en los fenómenos cuánticos observados en la naturaleza que busca beneficiarse de fenómenos como la superposición de estados y el entrelazamiento cuántico. A diferencia de una computadora clásica, una computadora cuántica es un sistema probabilístico, lo que la hace más parecida a una computadora clásica probabilísitca pero que, a diferencia de ésta, puede existir en más de un estado. De manera breve, la ejecución de una computadora cuántica consiste de tres pasos: preparación del estado inicial, implementación de una transformación lineal unitaria y medición del estado final.

3.1. Qubits

El qubit (quantum bit) es una entidad matemática que representa a un sistema cuántico de dos niveles y que corresponde a la unidad básica de información de una computadora cuántica. Al igual que en el cómputo clásico, el qubit es un sistema de dos estados, denotados como $|0\rangle$ y $|1\rangle$, pero, a diferencia de éste, puede existir también como una combinación lineal de ambos.

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

A esta combinación de estados se le conoce como superposición, donde α y β son números complejos que corresponden a las amplitudes de los estados $|0\rangle$ y $|1\rangle$, respectivamente, y que, además, cumplen con la condición de normalización de todo sistema cuántico $|\alpha|^2 + |\beta|^2 = 1$. En otras palabras, un qubit $|\psi\rangle$ es un elemento de norma unitaria en un espacio complejo de

Hilbert $\mathcal{H}^2$. $|0\rangle$ y $|1\rangle$ forman una base ortonormal de $\mathcal{H}^2$ conocida como la base computacional.

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \qquad\qquad |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

Aunque un qubit $|\psi\rangle$ puede existir en una superposición de $|0\rangle$ y $|1\rangle$, no es posible extraer información directamente de él, es necesario realizar una medición del qubit, lo que dará como resultado $|0\rangle$ con probabilidad $|\alpha|^2$ y $|1\rangle$ con probabilidad $|\beta|^2$. Pero, debido a la naturaleza cuántica del sistema, las mediciones alteran al sistema haciendo que colapse al estado medido, por lo que mediciones posteriores del qubit darán, con probabilidad igual a 1, el mismo resultado.[1].

Siguiendo la misma idea, y debido a que los sistemas cuánticos están descritos por un espacio de Hilbert, tenemos que un sistema de n qubits está representado por:

$$|\psi\rangle = \sum_{i_{n-1}=0}^{1} \dots \sum_{i_0=0}^{1} \alpha_{i_{n-1}\dots i_0} |i_{n-1}\rangle \otimes \dots \otimes |i_0\rangle = \sum_{i=0}^{2^n-1} \alpha_i |i\rangle \tag{3-2}$$

Donde $|i\rangle$ es el i-ésimo estado, $i = i_0 2^0 + \dots + i_{n-1} 2^{n-1}$ con i_k es el k-ésimo valor de la cadena binaria , con la restricción de que:

$$\sum_{i=0}^{2^n-1} |\alpha_i|^2 = 1 \tag{3-3}$$

Uno de los fenómenos de mayor interés es el entrelazamiento cuántico, el cual se presenta únicamente en sistemas con más de un qubit. Decimos que un sistema está entrelazado cuando no es separable, es decir, no es posible escribirlo de la forma:

$$|\psi\rangle = |\psi_{n-1}\rangle \otimes \dots \otimes |\psi_0\rangle \tag{3-4}$$

Con $|\psi_{n-1}\rangle, \dots, |\psi_0\rangle$ las funciones de estado de cada uno de los n subsistemas de los que se compone $|\psi\rangle$.

[1]Considerando que el sistema no evoluciona.

3.1.1. Esfera de Bloch

Debido a la condición de normalización del sistema podemos reescribir la ecuación 3-1 de la siguiente manera:

$$|\psi\rangle = e^{i\gamma}(cos\left(\frac{\theta}{2}\right)|0\rangle + e^{i\phi}sin\left(\frac{\theta}{2}\right)|1\rangle) \tag{3-5}$$

Con γ, ϕ y θ números reales donde, además, se puede omitir el término asociado a la fase global del sistema $e^{i\gamma}$ debido a que no tiene efectos observables. Simplificando la ecuación 3-5:

$$|\psi\rangle = cos\left(\frac{\theta}{2}\right)|0\rangle + e^{i\phi}sin\left(\frac{\theta}{2}\right)|1\rangle \tag{3-6}$$

La ecuación 3-6 proporciona una representación geométrica de un qubit, conocida como la esfera de Bloch, donde θ y ϕ representan puntos en la superficie de una esfera tridimensional.

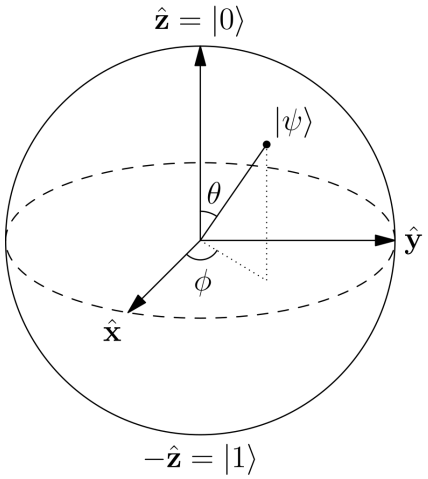

Figura 3-1: Esfera de Bloch.

La esfera de Bloch es una herramienta útil para entender el efecto de diversas transformaciones que se aplican a un qubit, aunque la extensión para más de un qubit no es trivial. Por ejemplo, la siguiente transformación intercambia las amplitudes de los estados $|0\rangle$ y $|1\rangle$, aunque podemos visualizarla como una rotación alrededor del eje X de π radianes.

$$\sigma_x = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

$$\sigma_x |\psi\rangle = e^{i\phi} sin\left(\frac{\theta}{2}\right)|0\rangle + cos\left(\frac{\theta}{2}\right)|1\rangle$$

3.2. Modelo de circuitos cuánticos

Una computadora cuántica puede ser visualizada como un conjunto finito de n qubits, o registro cuántico de n qubits en analogía con el caso clásico, a los cuales se les aplican una serie de transformaciones lineales, o compuertas lógicas cuánticas. Esta manera de visualizar a las computadoras cuánticas presenta el beneficio de que podemos utilizar ciertas ideas desarrolladas para el cómputo clásico.

3.2.1. Compuertas lógicas cuánticas de 1 qubit

Debido a que un sistema cuántico debe preservar la condición de normalización, todas las operaciones sobre un qubit están descritas por matrices unitarias M de 2×2.

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

$$Phase - Shift = R_\phi = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\phi} \end{bmatrix}$$

Es posible probar que estas dos transformaciones son suficientes para representar cualquier transformación lineal sobre la esfera de Bloch. De hecho, cualquier transformación unitaria cuyo efecto neto sea rotar el sistema de (θ_1, ϕ_1) a (θ_2, ϕ_2) se puede escribir como:

$$U = R_{\pi/2+\phi_2} H R_{\theta_2-\theta_1} H R_{-\pi/2-\phi_1} \tag{3-7}$$

En el modelo de circuitos cuánticos, estas compuertas se representan de la siguiente manera:

Figura 3-2: Representación gráfica de las compuertas $Hadamard$ y $Phase-shift$ en el modelo de circuitos cuánticos.

3.2.2. Compuertas lógicas cuánticas controladas

Como anteriormente se mencionó, una de las propiedades de mayor interés del cómputo cuántico es el entrelazamiento cuántico.

$$|\psi\rangle = |\psi_{n-1}\rangle \otimes \ldots \otimes |\psi_0\rangle$$

$$\left|\psi'\right\rangle = U_{n-1}\,|\psi_{n-1}\rangle \otimes \ldots \otimes U_0\,|\psi_0\rangle = \left|\psi'_{n-1}\right\rangle \otimes \ldots \otimes \left|\psi'_0\right\rangle$$

Es fácil ver que, cualquier número de operaciones de un qubit producen un sistema separable, es decir, no generan entrelazamiento. Por esta razón las compuertas de más de un qubit son de especial interés.

$$Controlled - Not = CNOT = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Por ejemplo, la compuerta $CNOT$, es una transformación lineal que toma dos qubits, denominados qubit control y qubit objetivo, la cual aplica la operación NOT sí y sólo sí el qubit control es $|1\rangle$. Por lo que, considerando el primer qubit como el control y el segundo como el objetivo, CNOT tiene el siguiente mapeo:

$$|00\rangle \mapsto |00\rangle \qquad |01\rangle \mapsto |01\rangle \qquad |10\rangle \mapsto |11\rangle \qquad |11\rangle \mapsto |10\rangle$$

Es sencillo ver que CNOT puede generar entrelazamiento entre dos estados:

$$|\psi\rangle = (\alpha\,|0\rangle + \beta\,|1\rangle)\,|0\rangle$$

$$CNOT\,|\psi\rangle = CNOT(\alpha\,|00\rangle + \beta\,|10\rangle)$$

$$CNOT\,|\psi\rangle = \alpha\,|00\rangle + \beta\,|11\rangle$$

De donde se puede ver que no es posible escribir $\alpha\,|00\rangle + \beta\,|11\rangle$ como en la ecuación 3-4. En general, todas las operaciones de un qubit se pueden transformar en operaciones controladas por un segundo qubit de manera sencilla.

$$U = \begin{bmatrix} \alpha & \beta \\ \gamma & \delta \end{bmatrix} \qquad Controlled - U = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \alpha & \beta \\ 0 & 0 & \gamma & \delta \end{bmatrix}$$

Figura 3-3: Representación gráfica de las compuertas $CNOT$ y $Controlled - U$ en el modelo de circuitos cuánticos.

3.2.3. Extensión de una compuerta a n qubits

Una ventaja de el modelo de circuitos cuánticos es que nos permite manipular los qubits de manera similar a como se manipulan los bits. Por ejemplo, sea un registro de tres qubits, representado por $|\psi\rangle$, al cual se le quiere aplicar una transformación U al primer y al tercer qubit. Como $|\phi\rangle$ pertenece a un espacio de Hilbert, podemos escribir:

$$|\psi\rangle = |\psi_0\rangle \otimes |\psi_1\rangle \otimes |\psi_2\rangle$$

Entonces la transformación que produce el resultado deseado es,

$$U'\,|\psi\rangle = U\,|\psi_0\rangle \otimes I\,|\psi_1\rangle \otimes U\,|\psi_2\rangle$$

Es decir,

$$U' = U \otimes I \otimes U$$

En general, cualquier operación se puede extender a un espacio de n qubits mediante el producto tensorial:

$$U = U_0 \otimes U_1 \otimes \ldots \otimes U_{n-1} \otimes U_n \tag{3-8}$$

3.2.4. Mediciones

Como previamente se mencionó, las mediciones son operaciones irreversibles que permiten extraer información de la computadora cuántica. Todo algoritmo cuántico termina siempre con la medición del sistema. Dos principios son de principal interés en el caso de las mediciones:

Principio de la medición diferida. *Toda medición en un estadio intermedio del circuito cuántico puede ser retrasada hasta el final del circuito. Si el resultado de una medición es utilizado para realizar una operación clásica controlada, se puede reemplazar dicha operación por una operación cuántica controlada.*

Principio de la medición implícita. *Todos los qubits que no fueron medidos al final del circuito se consideran como si hubiesen sido medidos.*

Estos dos principios dan lugar a dos resultados importantes: el primero dice que es posible construir, no necesariamente de manera trivial, un circuito cuántico reversible de un circuito cuántico no reversible; el segundo enuncia que si se realiza un medición parcial del sistema, mediciones de los qubits restantes no alteran el resultado de las mediciones realizadas.

3.3. Cómputo cuántico universal

De manera análoga al caso clásico en el que se puede construir cualquier compuerta lógica a partir de un conjunto relativamente pequeño de compuertas lógicas, como AND, OR y NOT, es posible demostrar que cualquier transformación unitaria U en $\mathcal{H}^{\otimes 2n}$ se puede descomponer

utilizando únicamente H, R_ϕ y $CNOT$ [4]. Se debe notar que R_ϕ es una compuerta lógica continua, lo que implica que el conjunto de compuertas lógicas cuánticas H, R_ϕ y $CNOT$ no es un conjunto discreto, sin embargo, es posible aproximar cualquier transformación unitaria U, hasta un cierto error ϵ, utilizando únicamente H, $R_{\pi/8}$ y $CNOT$. Este resultado es de vital importancia para el trabajo aquí expuesto ya que, como se menciona en el siguiente capítulo, para que el algoritmo de LGP (Linear Genetic Programming) pueda funcionar adecuadamente se requiere de un conjunto de operaciones completo.

3.4. Complejidad computacional cuántica

Para evaluar la calidad de un algoritmo cuántico se suele utilizar el modelo de consultas (Quantum query complexity) [9]. En este modelo la calidad de un cierto programa está dada por el número de consultas que realiza a una determinada función $f : \{0,1\}^n \mapsto \{0,1\}$, usualmente llamada caja negra u oráculo, para responder alguna pregunta relacionada a f en lugar de el tiempo que tarda en realizar un determinado cómputo.

Usualmente, la complejidad de consultas de un algoritmo suele ser igual a la complejidad temporal ya que en muchos casos es posible construir de manera eficiente a el oráculo y al resto del circuito, además de permitir la comparación de algoritmos clásicos (probabilísiticos) contra algoritmos cuánticos.

Como en cómputo cuántico se requiere que las operaciones sean reversibles, un oráculo O_f que representa a f es una transformación unitaria que realiza el siguiente mapeo:

$$|x\rangle |y\rangle \mapsto |x\rangle |y \oplus f(x)\rangle \tag{3-9}$$

Este modelo ha sido de gran utilidad para comparar algoritmos clásicos y cuánticos, lo que a su vez ayudó a demostrar el poder del cómputo cuántico sobre el cómputo clásico.

3.5. Simulación

Es posible simular una computadora cuántica en una computadora clásica considerando que se debe pagar un precio exponencial para almacenar los n qubits y realizar las operaciones.

Para la simulación utilizada en este trabajo se representó al registro cuántico como un vector de 2^n números complejos de manera que:

$$\begin{bmatrix} \alpha_0 \\ \vdots \\ \alpha_{2^n-1} \end{bmatrix} \sim |\psi\rangle = \sum_{i=0}^{2^n-1} \alpha_i |i\rangle$$

Para aplicar las operaciones, en lugar de extender las operaciones al tamaño del registro como en la ecuación 3-8, se optó por un método más eficiente que permite mantener las operaciones como matrices de m qubits, ya que la extensión a una matriz de n qubits requiere de 2^{2^n} coeficientes. El método consiste en generar máscaras de qubits para los diversos estados y aplicar las operaciones a estados cuya máscara sea la misma. Por ejemplo:

$$(X \otimes I \otimes I) \times \begin{bmatrix} \alpha_{000} \\ \alpha_{001} \\ \alpha_{010} \\ \alpha_{011} \\ \alpha_{100} \\ \alpha_{101} \\ \alpha_{110} \\ \alpha_{111} \end{bmatrix} = \begin{bmatrix} X \times \begin{bmatrix} \alpha_{000} \\ \alpha_{001} \end{bmatrix} \\ X \times \begin{bmatrix} \alpha_{010} \\ \alpha_{011} \end{bmatrix} \\ X \times \begin{bmatrix} \alpha_{100} \\ \alpha_{101} \end{bmatrix} \\ X \times \begin{bmatrix} \alpha_{110} \\ \alpha_{111} \end{bmatrix} \end{bmatrix} = \begin{bmatrix} \alpha_{001} \\ \alpha_{000} \\ \alpha_{011} \\ \alpha_{001} \\ \alpha_{101} \\ \alpha_{100} \\ \alpha_{111} \\ \alpha_{110} \end{bmatrix}$$

La máscara de un estado sólo depende del número binario asociado al estado y qubit al que se le va a aplicar la operación. Por ejemplo, si se desea aplicar una operación al segundo qubit, los estados $|101\rangle$ y $|111\rangle$ tienen una máscara de valor 4. La máscara no es más que la

transformación del valor binario del estado a entero, omitiendo el qubit o qubits a los cuales se les va a aplicar la operación.

Por otro lado, debido a que varias transformaciones sólo se pueden aproximar hasta una cierta precisión en una computadora clásica, las transformaciones que se aplican al sistema son no unitarias, por lo que para poder obtener una simulación correcta de una computadora cuántica es preciso minimizar la propagación de estos errores en la medida de lo posible. Para ello tras cada operación U aplicada al registro cuántico se recalcula y aplica el coeficiente de normalización del sistema. Gracias a la manera en la que se calculan dichas transformaciones, es posible recalcular el coeficiente de normalización mientras se computa una transformación U_i y aplicarlo mientras se computa la siguiente transformación U_{i+1}.

Otro aspecto importante que se incluyó en la simulación es la capacidad de realizar operaciones de ramificación (branching) según el resultado de la medición de un qubit i. Cada que el simulador inicia una operación de ramificación se termina la simulación actual y se inician dos simulaciones más, una en la que en la que el resultado de medir el i-ésimo qubit es $|0\rangle$ y otra en la que el resultado es $|1\rangle$. Cuando alguno de los eventos es muy poco probable, es decir, la probabilidad de que ocurra es inferior a un cierto umbral la simulación de ese caso es abortada para ahorrar tiempo de computo en resultados muy poco probables. De esta manera es posible registrar el comportamiento de ciertos algoritmos sin tener que realizar múltiples ejecuciones del mismo.

La simulación está dotada de una función Oráculo O_f más flexible de la que se expuso anteriormente. En primer lugar, O_f toma valores de una tabla de verdad para cada uno de los casos. Además, O_f fue construida de manera que puede elegir como tomar $|x\rangle\,|y\rangle$, es decir, O_f puede recibir el índice de los qubits del sistema para elegir de que manera aplicar la transformación, donde el último qubit de la lista es el qubit objetivo. Por ejemplo:

$$O_f(2,0,1) : |i_2 i_1 i_0\rangle \mapsto |i_2\rangle\,|i_1 \oplus f(x)\rangle\,|i_0\rangle \tag{3-10}$$

En general, O_f es una función de un conjunto de índices $I = \{i_a,\ \ldots,\ i_k,\ i_t\}$, donde

$x = i_a \dots i_k$, y un entero C, que índica la información cargada en el oráculo, que aplica la transformación:

$$O_f(I, C) : |i_0 \dots i_t \dots i_n\rangle \mapsto |i_0 \dots i_{t-1}\rangle \, |i_t \oplus f_C(x)\rangle \, |i_{t+1} \dots i_n\rangle \tag{3-11}$$

Finalmente, cabe resaltar que la simulación de la computadora cuántica está aunada al algoritmo LGP para facilitar la comunicación entre el algoritmo LGP y la simulación, así como para dotar de una mayor libertad a los programas generados mediante el algoritmo LGP.

Programación Genética Lineal

4.1. Programación Genética

La programación genética es un tipo de algoritmo evolutivo, basado en el algoritmo genético e inspirado en la selección natural, en el cual un conjunto de programas son modificados mediante una serie de operadores hasta encontrar una respuesta a un problema determinado.

Sea $f : X \mapsto Y$ una función que representa el problema que se desea resolver. Entonces, el objetivo de programación genética consiste en realizar una búsqueda de un programa o algoritmo que mejor mapee el subconjunto X de f al subconjunto de Y correspondiente. Normalmente, ésta búsqueda se realiza mediante una función auxiliar f', que está relacionada con f, que permite remplantear el problema como una maximización o minimización de algunas variables. Coloquialmente, a f' se le da el nombre de función objetivo o fitness.

Dado un lenguaje L, compuesto por un conjunto de instrucciones y terminales, se define el espacio de genotipos G como aquel que contiene todos los posibles programas que se pueden construir con los elementos de L y el espacio de fenotipos P como el conjunto de todas las funciones $F_{gp} : I^n \mapsto O^m$, con I el dominio del espacio del problema y O el codominio, tales que F_{gp} es expresada por un programa gp de G. Entonces, la función objetivo $F : P \mapsto V$ es una medida de la calidad de la predicción. Normalmente, $V = \mathbb{R}^+$ para problemas continuos y $\mathbb{N}$ para problemas discretos y F es una función de error con respecto de la función o modelo deseado.

La expresividad de un lenguaje L está dada por los elementos de los conjuntos de instrucciones y terminales. Para poder resolver cualquier problema mediante programación genética, es necesario garantizar que el lenguaje L es lo suficientemente expresivo como para poder representar un programa óptimo, o al menos aproximarlo en un intervalo del dominio del problema. Por otro lado, un lenguaje L con elementos innecesarios o redundantes puede incrementar drásticamente la dificultad de la búsqueda. Debido a que, a priori, no se conoce la forma del programa óptimo, determinar el mejor conjunto L no es trivial, aunque si se garantiza que L sea un lenguaje Turing-completo es posible encontrar un programa adecuado, no obstante, un lenguaje Turing-completo requiere de ciclos infinitos y como, a priori, no es posible determinar si un programa va a terminar es necesario imponer límites sobre la extensión de los programas.

En programación genética existen dos tipos de operadores: operadores de selección y operadores de variación. Sea $P(t) \subset G$ la población al tiempo t y P' un subconjunto de $P(t)$ tal que $|P'| = n$, un operador de selección $s : G_n \times P_n \mapsto G_\mu$ es una función que selecciona $\mu < n$ individuos de P'. Un operador de variación $v : G_\mu \mapsto G_\lambda$ crea λ programas a partir de μ programas. Comúnmente, cuando $\mu = \lambda = 2$ se denomina al operador de variación como operador de cruza y cuando $\mu = \lambda = 1$ como operador de mutación. Los operadores de variación deben garantizar que todos los programas generados sean sintácticamente correctores.

4.1.1. Algoritmo

A continuación se muestra un algoritmo de referencia de programación genética:

A) Crear una población aleatoria de N programas sintácticamente correctos.

B) Evaluar a los individuos N de la población.

C) Seleccionar a un individuo de la población.

D) Modificar al individuo seleccionado con uno o más operadores de variación.

E) Agregar al individuo modificado a la nueva población.

F) Repetir C, D, E hasta que la nueva población tenga tamaño N.

G) Regresar a B hasta que se cumpla el criterio de paro.

H) Evaluar los programas con los datos de validación y seleccionar al mejor.

Aunque el algoritmo anterior es un algoritmo generacional, es decir, realiza iteraciones de una población, también existen alternativas en las que los individuos son reinsertados dentro de la misma población; estos son conocidos como algoritmos estacionarios. En la literatura se pueden encontrar diversas variaciones del algoritmo con el objetivo de aumentar el poder de resolución del mismo, como en el caso del algoritmo utilizado en el presente trabajo.

4.2. Programación Genética Lineal

Con la popularización de la programación genética surgieron nuevas propuestas para la representación de los programas en contraste con la forma clásica, en la que los programas o algoritmos son representados mediante gráficas de árboles, donde los nodos interiores representa un elemento de un conjunto de instrucciones y las hojas es un elemento de un conjunto de terminales. Una de estas propuesta es la representación lineal.

4.2.1. Representación

La programación genética lineal [15], comúnmente abreviada LGP, está basada en el paradigma de programación imperativa, en el cual los programas se visualizan como una serie de instrucciones las cuales deben ser ejecutadas ordenadamente, de ahí el nombre de programación genética lineal.

De manera similar a la arquitectura de Von Neumann, en esta variante de programación genética, los programas consisten de una serie de instrucciones u operaciones sobre un conjunto de registros. A cada instrucción de un programa imperativo se le denomina instrucción imperativa. Una instrucción imperativa consiste de una operación, un conjunto de registros sobre los cuales se va a realizar la operación, normalmente dos, y un registro de asignación, donde

se almacena el resultado de dicha operación; aunque, es posible definir una instrucción sobre n operaciones y m registros, esto no necesariamente aumenta la expresividad de un programa.

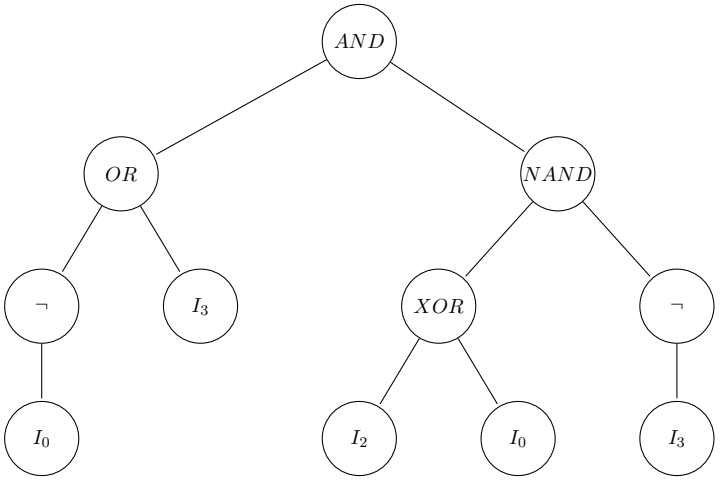

Figura 4-1: Ejemplo de un programa GP.

En LGP, un programa consiste de una cantidad variable de instrucciones imperativas de la forma $< op, i, j, k >$ donde op es una operación del conjunto de instrucciones de L e i, j y k son índices que corresponden a algún registro. Usualmente las constantes se introducen al sistema mediante una serie de registros protegidos contra escritura ya que, el mismo sistema permite la evolución de diferentes constantes mediante las operaciones del conjunto de instrucciones. Vale la pena resaltar que esta codificación de las instrucciones permite codificar a los programas como matrices de cuatro columnas por n renglones.

Debido a que en LGP un programa opera sobre una serie de registros, es necesario definir una cantidad adecuada de registros sobre los que va a operar el programa. Una cantidad pequeña de registros ocasionarán que los programas estén sobreescribiendo información de manera constante. Por otro lado, un número grande de registros incrementará el tamaño del espacio de búsqueda de manera considerable. También es necesario definir qué registros serán utilizados para la entrada de datos y cuales para la salida.

4.2.2. Instrucciones efectivas y no efectivas

Debido a la estructura de los programas en LGP es común que se generen estructuras redundantes para la solución del programa. En analogía a lo que ocurre en la naturaleza con los segmentos de ADN que no son expresados al momento de transcribir un gen, a estas regiones redundantes del programa se les da el nombre de intrones.

Definición Se define a una instrucción como efectiva en la posición p si dicha instrucción tiene una repercusión en el resultado de un programa para por lo menos un elemento del dominio del problema. Una instrucción no efectiva o intrón es aquella que no modifica el resultado del programa en ninguna situación.

Definición Se define a un registro como efectivo en la posición p si su manipulación tiene una repercusión en el resultado de un programa para por lo menos un elemento del dominio del problema.

Las definiciones anteriores sugieren una clasificación de los intrones en dos tipos: intrones estructurales e intrones semánticos.

Definición Se dice que un intrón es un intrón estructural si es el resultado de instrucciones que manipulan registros no efectivos.

Definición Se le llama intrón semántico a aquel que surge de la manipulación de registros efectivos.

Para reducir los tiempos de evaluación en LGP, el paso limitante de esta técnica, es posible ignorar todos los intrones al momento de la ejecución del programa, consiguiendo mejoras considerables en los tiempos de ejecución. Normalmente, las instrucciones no efectivas son marcadas e ignoradas ya que eliminar dichas instrucciones puede ser perjudicial para el proceso de evolución. Este proceso sólo suele realizarse sobre intrones estructurales ya que detectar intrones semánticos es un problema no trivial y posiblemente más costoso que la ejecución del programa completo. La eliminación de intrones semánticos puede ser útil al final de la ejecución de un

algoritmo LGP para facilitar la interpretación del programa respuesta. Es posible reducir la cantidad de intrones semánticos evitando su creación al momento de crear las instrucciones o aplicar ciertos operadores.

4.2.3. Representación gráfica

Como anteriormente se mencionó, en la codificación tradicional de programación genética un programa se puede visualizar como un gráfica de árbol en donde los nodos representan elementos del conjunto de operaciones y las hojas elementos del conjunto de terminales.

De manera análoga, un programa de LGP corresponde a una gráfica dirigida. La gráfica es acíclica si el conjunto de operaciones no contiene operaciones bucle como la operación for. Cada instrucción corresponde a un nodo de la gráfica y cada arista dirigida que sale del nodo corresponde a cada uno de los operandos de la instrucción. Los nodos de los que no sale ninguna arista, conocidos como sumideros, son los elementos del conjunto de terminales.

El código no efectivo de un programa se puede visualizar como elementos de la gráfica a los que ningún nodo efectivo apunta aunque no necesariamente disconexos, ya que puede que un nodo no efectivo apunte a un nodo efectivo.

```
double Programa (double[8] registro_inicial)
{
        double r[8] = registro_inicial;

        r[0] = r[1] + r[2];
        r[4] = r[3] / r[7];
        r[0] = r[6] + r[2];
        if (r[0] < 3.14)
                r[0] = r[7] * r[2];
        r[3] = r[2] - r[4];
        r[0] = r[5] + 1.46;
        if (r[3] > r[4])
                if (r[3] < r[2])
                        r[4] = r[1] - r[2];
        r[1] = r[1] * r[4];
        r[1] = r[7] + 9.15;

        return r[3];
};
```

Figura 4-2: Representación gráfica de un programa lineal.

4.3. Función de fitness

La función objetivo o función de fitness es una medida de la calidad de los programas. Es común referirse al fitness de un programa como el resultado de la evaluación de dicho programa con la función objetivo en todo el dominio de entrenamiento.

La selección de la función objetivo adecuada es crucial para la calidad de los resultados de un algoritmo de programación genética. Incluso cambios, en principio, insignificantes de la función de fitness pueden resultar críticos en la obtención de una buena respuesta ya que ésta es la principal responsable de dirigir la búsqueda en el espacio de programas.

$$f = \sum_{i}^{N} (F_{gp}(I_i) - O_i)^2 \tag{4-1}$$

Una de las funciones de fitness más utilizadas, para problemas continuos, es la función de error con respecto del modelo ideal (ecuación 4-1) que se está buscando y diversas variantes

de ésta como la función de error normalizada, etc. Para problemas discretos suelen contarse la cantidad de aciertos y de errores, que es equivalente de la función de error para dominios discretos.

4.4. Operadores

En conjunto con la función de fitness, los operadores de selección y operadores de variación son los encargados de dirigir la búsqueda en el espacio de programas. Normalmente, tras seleccionar a un individuo para que su información pase al siguiente estadio se suele aplicar uno o más operadores de variación con una cierta probabilidad.

No existe un conjunto de operadores definitivo, puesto que los operadores que resultan eficaces en un problema pueden resultar inservibles en otro y viceversa. Estudiar las carencias de los programas producidos por un algoritmo de LGP e intentar corregirlas es una manera de dar con los mejores operadores para un problema.

4.4.1. Selección

Los dos operadores de selección más usados son: selección por Torneo y selección por Ruleta. En la selección por Torneo se seleccionan n individuos de la población de manera aleatoria y de éstos se selecciona al individuo con el mejor fitness. En la selección por ruleta, se genera un número aleatorio entre 0 y la suma del fitness de todos los individuos, posteriormente se suma el fitness individuo a individuo hasta que la suma acumulada sea igual o mayor al número aleatorio; de esta manera, los individuos con mejor fitness tendrán mayor posibilidad de ser seleccionados y viceversa.

Aunque ambos operadores son viables, se suele utilizar el operador por Torneo, o alguna variante de este, con mayor frecuencia debido a que suele inducir una menor presión en la población y con ello prevenir estancamientos prematuros en mínimos locales. Además de estos dos, es posible definir otros operadores de selección e incluso hacerlos sensibles al problema a tratar.

4.4.2. Macro variaciones y micro variaciones

Las macro variaciones son aquellas variaciones que operan al nivel de instrucciones. Este tipo de variaciones son las encargadas de realizar la búsqueda global dentro de los algoritmos de GP. En LGP existen diversos operadores de macro variaciones, pero los más utilizados son: cruza, inserción y eliminación.

El operador tradicional de cruza toma dos individuos, selecciona dos puntos de cruza de manera aleatoria en ambos individuos e intercambia todas las instrucciones comprendidas entre los dos puntos de cruza entre los dos individuos, dando lugar a dos nuevos individuos. Existen diversas variaciones de este operador: sensibles al tamaño de la información intercambiada, sensibles a la posición de la información intercambiada, cruzas que sólo producen un único individuo, etc.

Los operadores de inserción y eliminación, funcionan de manera similar, insertando o eliminando una instrucción en una posición arbitraria del individuo.

Las micro variaciones operan al nivel de los componentes de la instrucción y son las encargadas de realizar la búsqueda local en el espacio de programas. En LGP solamente existen dos tipos: cambio de operación y cambio de registros. Estos operadores actúan remplazando alguno de los cuatro números de los que está compuesta una instrucción por algún otro número que produzca un individuo sintácticamente correcto.

4.5. Otras modificaciones

A continuación se presentan brevemente las modificaciones realizadas al algoritmo que se utilizaron para aumentar su poder de resolución. Al final de esta sección se presenta el algoritmo LGP utilizado para resolver los problemas planteados en el presente trabajo.

4.5.1. Elitismo

El elitismo es un mecanismo que permite la preservación de los mejores individuos encontrados por el algoritmo hasta la iteración actual. El proceso consiste en copiar a los mejores n individuos encontrados hasta el momento a la siguiente población.

4.5.2. Subpoblaciones (Demes)

El esquema de subpolaciones consiste en la evolución paralela de más de una población. En este esquema las poblaciones evolucionan de manera aislada la una de la otra, lo que permite una exploración simultánea de diversas regiones del espacio de programas generado por L. Debido a que los procesadores modernos permiten la ejecución de múltiples procesos de manera paralela, normalmente se permite que los saltos generacionales de cada subpoblación sean independientes.

Aunque las subpoblaciones pueden mantenerse completamente aisladas, es frecuente permitir que las subpoblaciones se comuniquen entre ellas cada cierto número de generaciones g. A este proceso se le conoce comúnmente como migración y se da entre dos subpoblaciones cuando una de ellas alcanza el umbral de migración g. El proceso consiste en seleccionar a n individuos y a otra subpoblación; posteriormente la subpoblación receptora, al momento de crear la población de la siguiente generación, introduce a los individuos de la población emisora sin realizar ningún tipo de modificación de los mismos.

4.5.3. ALPS

La técnica de ALPS (Age-Layered Population Structure) [12] consiste en asignarle a cada individuo una edad, la cual es representativa de cuanto tiempo lleva evolucionando la información que porta dentro de la población. Con ello, la población es dividida en una serie de capas que sólo pueden ser pobladas con individuos cuya edad se encuentre dentro de un cierto rango definido para cada capa. Los límites para cada capa son calculados mediante alguna sucesión conveniente, como $f_n = n^2$, multiplicada por una constante o múltiplo de capa. La capa supe-

rior no posee límite de edad y la capa inferior introduce nuevos individuos de manera periódica. Los individuos solamente compiten contra individuos dentro de su misma capa y el operador de cruza sólo puede seleccionar individuos de la misma capa o de la capa inmediatamente inferior.

La edad de los individuos es asignada de la siguiente manera:

- Todos los programas inician con edad cero.
- Cada que un individuo participa en una operación de cruza su edad es aumentada en uno; si el individuo participa en más de una cruza su edad no vuelve a aumentar.
- Cada que a un individuo se le aplica un operador de variación, diferente a la cruza, su edad aumenta en uno; si al individuo se le aplica más de un operador de variación su edad no vuelve a aumentar.

Cada generación y antes de la creación de la nueva población, los individuos que superan la edad de la capa que habitan intentarán eliminar a otro individuo de la capa inmediatamente superior que tenga un fitness menor al del individuo. En caso de no conseguirlo el individuo es eliminado de la población.

4.5.4. Algoritmo

El algoritmo LGP utilizado en este trabajo de investigación es el siguiente:

A) Crear P poblaciones aleatorias de N programas sintácticamente correctos separados en l capas.

B) Evaluar a los N individuos de cada población.

C) Revisar si se cumplió la condición de paro.

D) En caso de que se alcance el umbral de migración g, enviar m individuos a una población arbitraria.

E) Crear a la nueva población.

1) Mover a los individuos necesarios a la siguiente capa o eliminarlos de la población
2) Añadir automáticamente a los mejores individuos de la población a la siguiente población.
3) Comprobar si no hay una población de migrantes en espera, en dado caso, añadirlos a la siguiente población.
4) Seleccionar a un individuo de la población mediante un torneo de k individuos.
5) Modificar al individuo seleccionado con uno o más operadores de variación con una cierta probabilidad; en caso de que la operador de variación sea un operador de cruza, seleccionar a un segundo padre mediante otro torneo de k individuos.
6) Agregar al individuo modificado a la nueva población.
7) Regresar a 4 hasta que la nueva población tenga tamaño N.

F) Regresar a B hasta que se cumpla el criterio de paro.

G) Seleccionar al mejor programa encontrado en cada población.

Experimentos

Con la intención de demostrar la capacidad del sistema propuesto en este trabajo para encontrar algoritmos viables se plantearon los siguientes problemas: identificación de oráculos balanceados y constantes, esto es, encontrar un algoritmo equivalente al algoritmo de Deutsch Josza; identificar un elemento de un base de datos no ordenada, de manera similar al algoritmo de Grover; y, finalmente, el diseño de algoritmos que evalúen cuatro funciones booleanas de AND y OR. Algunos de estos problemas también fueron resueltos por L. Spector y H. Barnum utilizando técnicas convencionales de GP [22, 23, 24, 2].

Debido a que los algoritmos LGP son limitados por que tan rápido pueden evaluar a los programas y, también, al incremento exponencial de recursos de la simulación del registro cuántico con el número de qubits, todos los problemas de este trabajo se realizaron utilizando un registro cuántico de tres qubits, aunque, en principio, el algoritmo LGP debería ser capaz de resolver problemas en sistemas más grandes.

Para todos los problemas del tratados se utilizó la siguiente función de fitness:

$$F = \sum_{i=0}^{N} P_u(p_{e_i}) + p_{e_{max}} \tag{5-1}$$

Donde i representa al i-ésimo caso del conjunto de evaluación de N elementos, p_{e_i} es la probabilidad de error del i-ésimo caso, P_u es una función de penalización cuya evaluación es

cero si p_{e_i} es menor a un cierto valor umbral u o igual a un valor C en cualquier otro caso y $p_{e_{max}}$ es la probabilidad máxima de error observada. Por lo que F es una función que primero prioriza la minimización general del error y que, posteriormente, procura minimizar únicamente la probabilidad máxima de error.

Número de ejecuciones:	20
Número de máximo de generaciones:	5000
Tamaño del registro cuántico:	3
Número de llamadas al oráculo permitidas:	1
Umbral de penalización:	0.48
Factor de penalización:	10
Número de Demes:	8
Número de individuos por capa:	100
Número de capas:	10
Múltiplo de capa:	25
Porcentaje de elitismo:	0.03
Generaciones entre migraciones:	75
Porcentaje de migrantes:	0.03
Tamaño del torneo:	7
Probabilidad de que el segundo padre se eliga de una capa inferior:	0.2

Tabla 5-1: Parámetros comunes en todas las ejecuciones del algoritmo LGP.

Para cada experimento se realizaron 20 repeticiones y se seleccionó al mejor algoritmo obtenido de entre todas las repeticiones, considerando una relación fitness contra complejidad de la solución. Posteriormente se realizó una simplificación manual de los algoritmos en los casos que se consideró necesario.

5.1. Algoritmo de Deutsch Josza

El algoritmo de Deutsch Josza [10], aunque de poca utilidad práctica, fue elegido debido a su importancia histórica, puesto que fue el primer algoritmo en el que se observó que una computadora cuántica se desempeña mejor que una computadora clásica.

Sea un oráculo O_f que evalúa una función $f : \{0,1\}^m \mapsto \{0,1\}$ de la cual únicamente se sabe que es constante o balanceada, es decir, que exactamente la mitad de sus evaluaciones regresan cero y la otra mitad uno. El problema consiste en determinar cuándo la función f es constante o balanceada. Clásicamente, responder a esta pregunta requiere de por lo menos $N/2+1$ llamadas al oráculo en el peor de los casos; por otro lado, en cómputo cuántico, David Deutsch y Richard Jozsa mostraron que basta con una llamada al oráculo para responder a esta pregunta.

Probabilidad de cruza:	0.5
Probabilidad de micro mutación:	0.35
Probabilidad de macro mutación:	0.15
Tamaño mínimo de los programas:	4
Tamaño máximo de los programas:	12
Funciones:	O_f, S_x, H, P, $R_{\pi/8}$, $CNOT$

Tabla 5-2: Parámetros de ejecución del algoritmo LGP para resolver el problema de Deutsch Josza.

La soluciones del problema son evaluadas con la probabilidad de medir el estado $|0\ldots0\rangle$ con probabilidad uno cuando f es constante y con probabilidad cero cuando f es balanceada [1]. Con la intención de recuperar un algoritmo de mayor parecido con el obtenido por Deutsch y Josza, no se permitió al oráculo O_f utilizar índices para definir x a voluntad.

[1] Debido a que solamente se desea medir una propiedad del sistema es, en principio, posible pedir al algoritmo LGP que responda utilizando cualesquiera dos estados del registro cuántico, pero se optó por seguir los resultados de la versión de la literatura en donde si f es constante se mide el estado $|0\ldots0\rangle$ con probabilidad uno y si es balanceada con probabilidad cero.

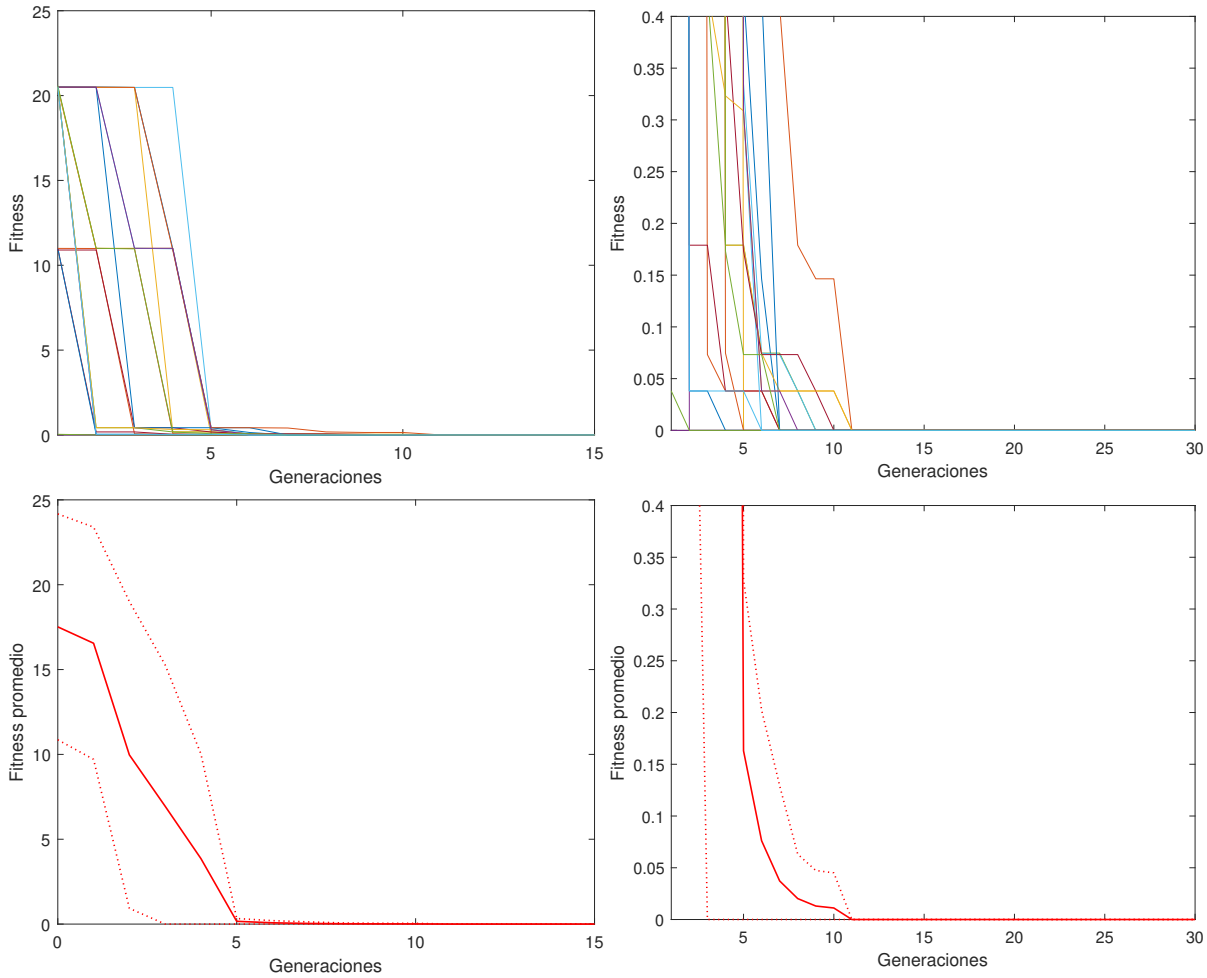

Figura 5-1: Fitness de los individuos a lo largo de diferentes generaciones. En la parte superior se muestra el fitness de cada una de las 20 corridas; en la parte inferior el fitness promedio de las 20 corridas, las líneas punteadas indican la desviación estándar.

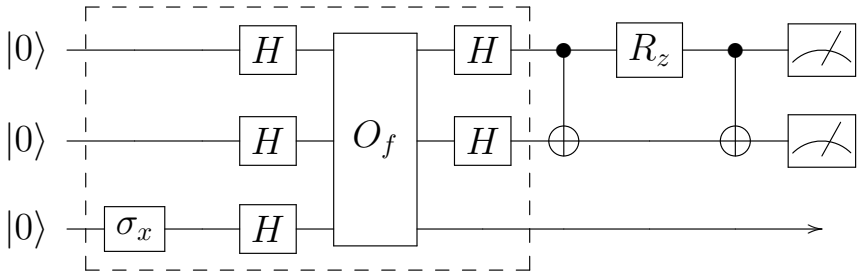

Figura 5-2: Mejor solución obtenida en una de las ejecuciones del algoritmo.

Para este problema, el sistema fue capaz de encontrar soluciones perfectas en cada una de las 20 corridas, tomándole en promedio 11 generaciones. En la figura 5-2 se muestra una de las soluciones encontradas por el algoritmo.

En este caso el algoritmo no sólo encontró una solución válida, sino que encontró, esencialmente, el algoritmo original (indicado en la figura mediante el recuadro punteado). Todo lo que está fuera del recuadro es irrelevante para el algoritmo ya que el efecto neto de esas compuertas es aplicarle una fase al estado si el primer qubit es uno:

$$CNOT \quad R_z \otimes I \quad CNOT \quad \alpha\,|00\rangle = \alpha\,|00\rangle$$

$$CNOT \quad R_z \otimes I \quad CNOT \quad \alpha\,|01\rangle = \alpha\,|01\rangle$$

$$CNOT \quad R_z \otimes I \quad CNOT \quad \alpha\,|10\rangle = \alpha'\,|10\rangle$$

$$CNOT \quad R_z \otimes I \quad CNOT \quad \alpha\,|11\rangle = \alpha'\,|11\rangle$$

5.2. Algoritmo de búsqueda de Grover

El segundo problema para evaluar la capacidad del sistema es el algoritmo de Grover. Este problema, al igual que el primero, es de gran importancia histórica.

El problema consiste en encontrar un elemento en una base de datos no ordenada de N elementos. En este problema revisar un elemento de la base de datos es equivalente a realizar una llamada al oráculo con el identificador x de dicho elemento. Un algoritmo clásico, en promedio, requiere revisar $N/2$ elementos de la base de datos y en el peor de los casos necesita revisar $N-1$ elementos para obtener la respuesta con certeza. Grover mostró que una computadora cuántica puede encontrar el elemento de la base de datos con $\sqrt{N}$ llamadas al oráculo[11].

El problema equivalente para un registro cuántico de 3 qubits es una búsqueda en una base de datos de 4 elementos, donde se espera una única llamada al oráculo para encontrar la

respuesta correcta. De manera similar y por la misma razón que en el caso anterior, la elección de x se mantiene fija al momento de aplicar el operador O_f.

Probabilidad de cruza:	0.5
Probabilidad de micro mutación:	0.35
Probabilidad de macro mutación:	0.15
Tamaño mínimo de los programas:	18
Tamaño máximo de los programas:	36
Funciones:	O_f, S_x, H, P, $R_{\pi/8}$, $CNOT$

Tabla 5-3: Parámetros de ejecución del algoritmo LGP para resolver el problema de Grover.

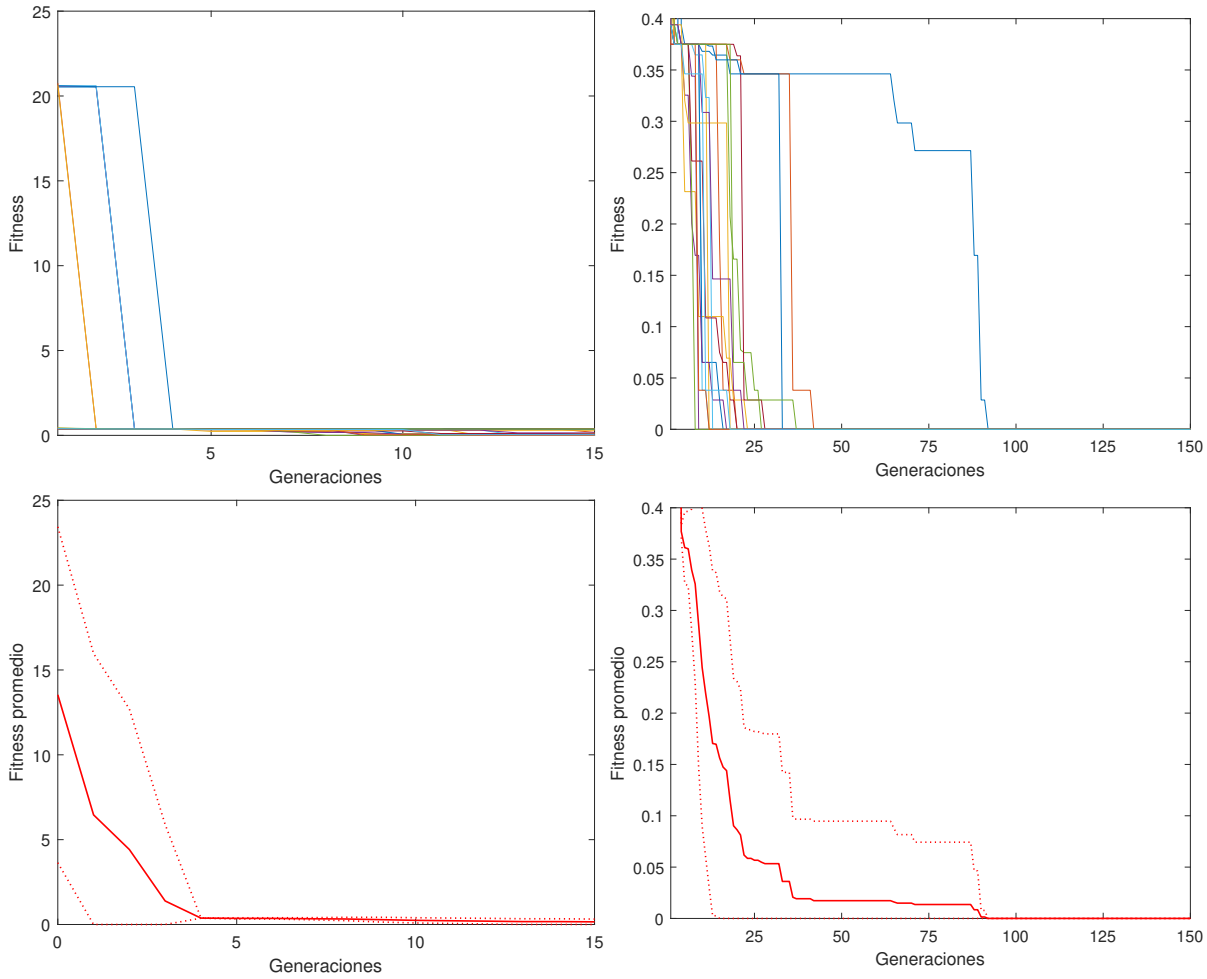

Figura 5-3: Fitness de los individuos a lo largo de diferentes generaciones. En la parte superior se muestra el fitness de cada una de las 20 corridas; en la parte inferior el fitness promedio de las 20 corridas, las líneas punteadas indican la desviación estándar.

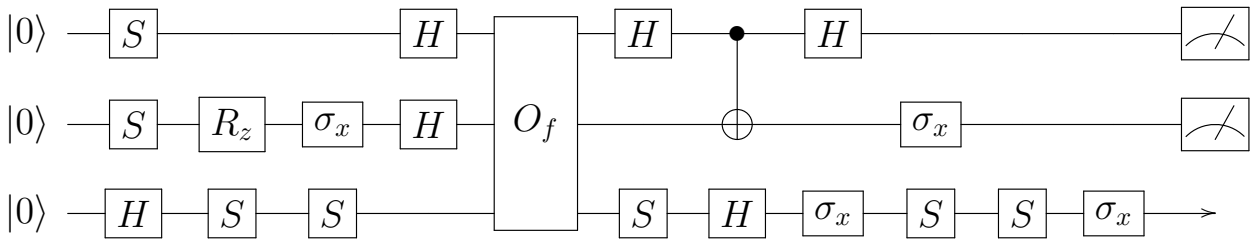

Figura 5-4: Mejor solución obtenida en una de las ejecuciones del algoritmo.

En este problema, el sistema encontró soluciones exactas en poco menos de 100 generaciones en cada una de las 20 corridas. A diferencia del problema anterior, los algoritmos encontrados por el sistema tenían una cantidad mucho mayor de compuertas redundantes, o intrones, como las primeras dos compuertas S en el primer y segundo qubit o todas las compuertas en el tercer qubit posteriores a la compuerta del oráculo O_f. Posteriormente a la eliminación de las redundancias y tomando en cuenta que $\sigma_x = SS$ y $H\sigma_z = \sigma_x H$, se puede reescribir el circuito de la figura 5-4 como el circuito de la figura 5-5.

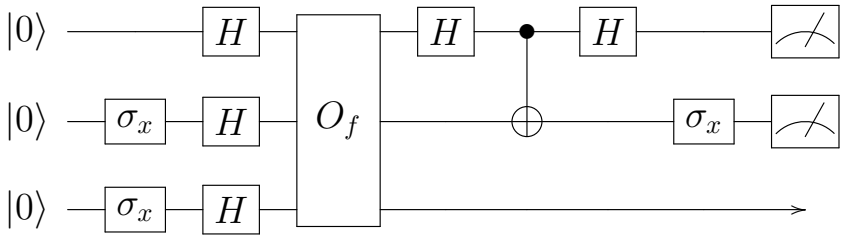

Figura 5-5: Versión simplificada del programa mejor programa obtenido.

El algoritmos simplificado tiene una cierta semblanza al algoritmo de Grover original. Comparándolo con el algoritmo de Grover, para una base de datos de cuatro elementos, es considerablemente más compacto, utilizando solamente dos tercios de las compuertas que utiliza la versión de Grover.

5.3. Funciones Booleanas de AND y OR

Finalmente, se probó el sistema para diseñar algoritmos capaces de evaluar funciones booleanas $B : \{0,1\}^4 \mapsto \{0,1\}$, compuestas únicamente por AND y OR, utilizando una única llamada al oráculo.

$$B_1 = (x_0 \vee x_1) \wedge (x_2 \vee x_3) \tag{5-2}$$

$$B_2 = (x_0 \wedge x_1) \vee (x_2 \wedge x_3) \tag{5-3}$$

$$B_3 = (x_0 \wedge x_1) \wedge (x_2 \vee x_3) \tag{5-4}$$

$$B_4 = (x_0 \wedge x_1) \vee (x_2 \vee x_3) \tag{5-5}$$

Santha probó que para cualquier función booleana $B(x_0, \ldots, x_N)$ en la que todas las variables x_i aparecen una única vez no es posible construir ningún algoritmo Monte Carlo con probabilidad de error máxima menor a p con menos de $(1 - sp)Q$ llamadas al oráculo, donde s indica si el error está acotado por un solo lado ($s = 1$) o por ambos lados ($s = 2$) y Q es el tiempo del algoritmo Las Vegas óptimo [19].

De lo anterior se sigue que la probabilidad de error máxima de un algoritmo probabilístico clásico óptimo que evalúa la función B y que sólo consulta al oráculo una vez está acotada por 1/3 si el error está acotado por ambos lados. Por lo tanto, cualquier algoritmo cuántico, cuya probabilidad de error máxima $p_{e_{max}}$, que cumpla que $p_{e_{max}} < 1/3$ será mejor que el mejor algoritmo probabilístico clásico correspondiente.

Probabilidad de cruza:	0.1
Probabilidad de micro mutación:	0.63
Probabilidad de macro mutación:	0.27
Tamaño mínimo de los programas:	12
Tamaño máximo de los programas:	28
Funciones:	O_f, S_x, H, P, $R_{\pi/8}$, $CNOT$, $SqrtNOT$, U_θ

Tabla 5-4: Parámetros de ejecución del algoritmo LGP para resolver el problema de funciones booleanas de $ANDs$ y ORs, sin mediciones parciales.

Debido a trabajos anteriores en esta área de los cuales parece inferirse la idea de que las mediciones parciales pueden mejorar la calidad de las soluciones cuando se utiliza esta técnica [23, 24, 2], también se estudió el efecto de las mediciones parciales al momento de encontrar soluciones viables mediante programación genética.

Probabilidad de cruza:	0.1
Probabilidad de micro mutación:	0.63
Probabilidad de macro mutación:	0.27
Tamaño mínimo de los programas:	12
Tamaño máximo de los programas:	28
Funciones:	O_f, If, $ElseEndIf$, S_x, H, P, $R_{\pi/8}$, $CNOT$, $SqrtNOT$, U_θ

Tabla 5-5: Parámetros de ejecución del algoritmo LGP para resolver el problema de funciones booleanas de $ANDs$ y ORs, utilizando mediciones parciales.

En esta sección solamente se presenta el proceso de evolución promedio de cada una de los algoritmos con su respectivo programa simplificado. Los datos de las 20 corridas y los algoritmos originales se pueden consultar en el apéndice B.

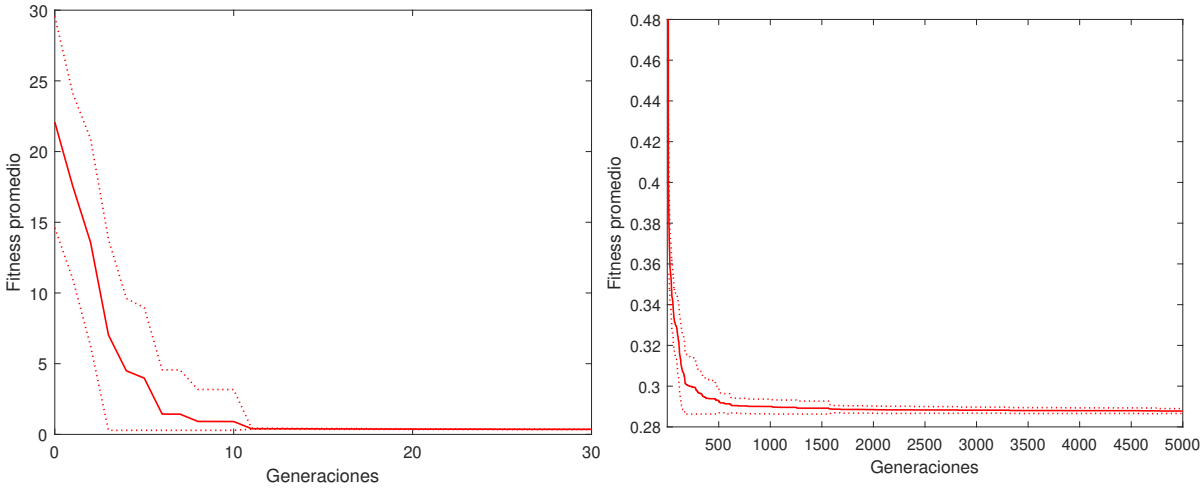

Figura 5-6: Fitness promedio de las 20 corridas a lo largo de diferentes generaciones para la función B_1, sin mediciones parciales, las líneas punteadas indican la desviación estándar.

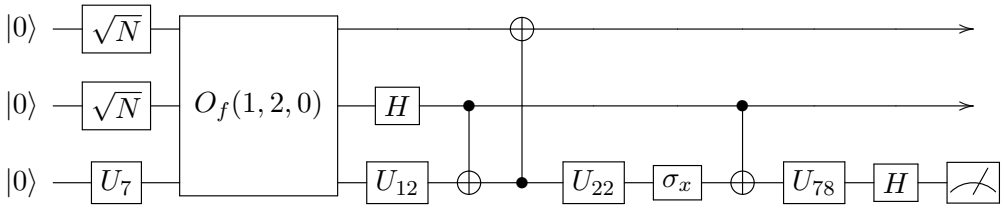

Figura 5-7: Versión simplificada del mejor programa obtenido para la función B_1, sin mediciones parciales ($p_{e_{max}} = 0.2879$). La medición final se realiza sobre el último qubit.

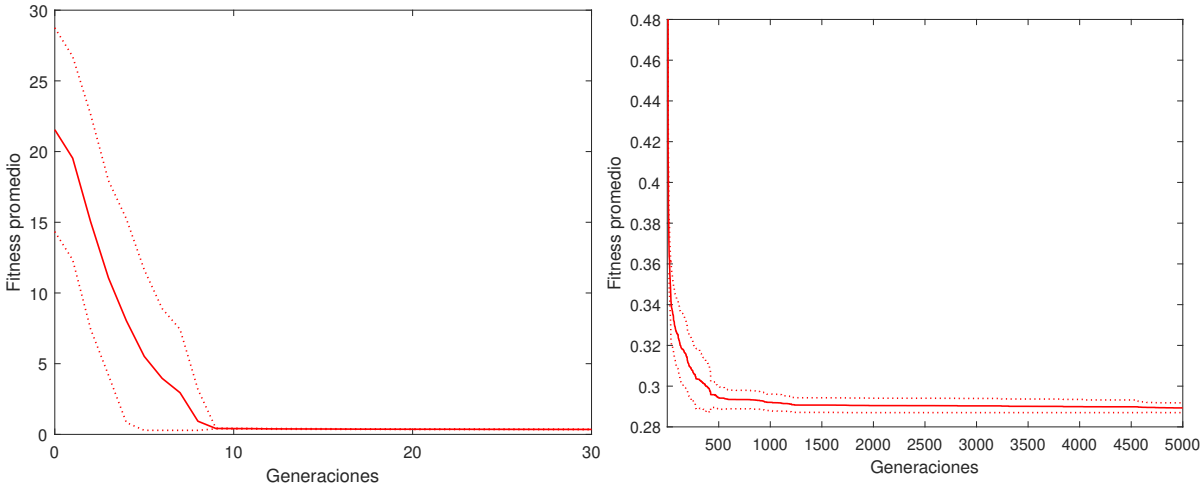

Figura 5-8: Fitness promedio de las 20 corridas a lo largo de diferentes generaciones para la función B_2, sin mediciones parciales, las líneas punteadas indican la desviación estándar.

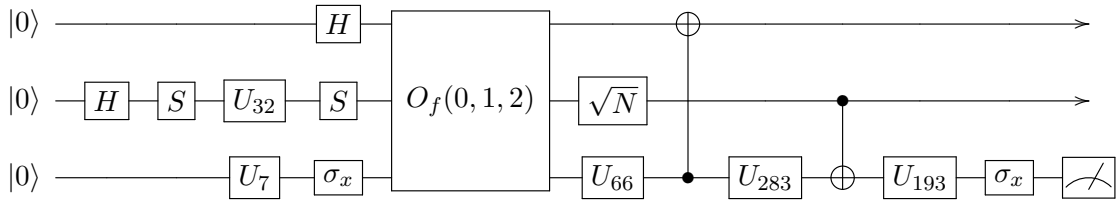

Figura 5-9: Versión simplificada del mejor programa obtenido para la función B_2, sin mediciones parciales ($p_{e_{max}} = 0.2891$). La medición final se realiza sobre el último qubit.

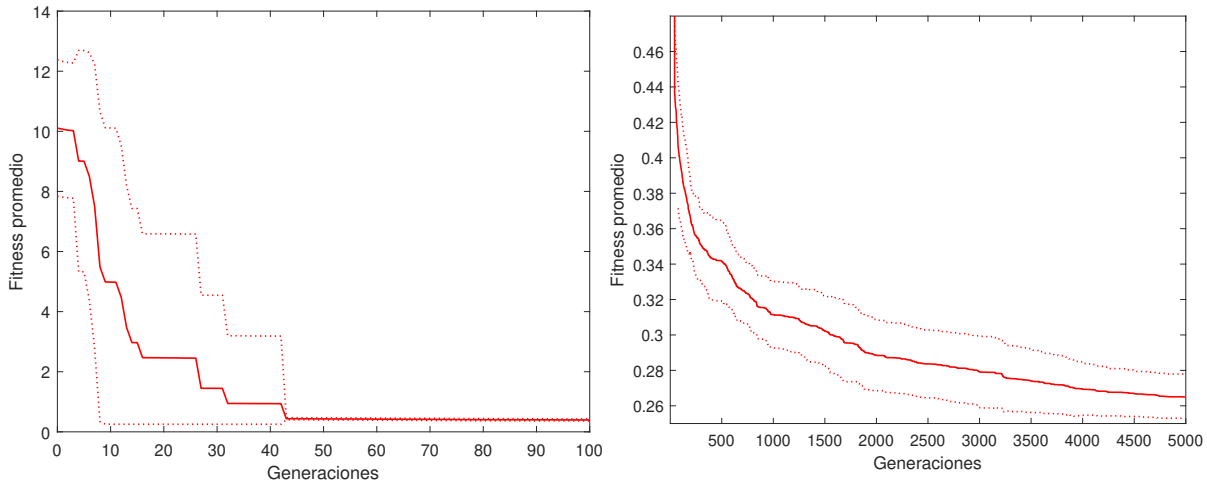

Figura 5-10: Fitness promedio de las 20 corridas a lo largo de diferentes generaciones para la función B_3, sin mediciones parciales, las líneas punteadas indican la desviación estándar.

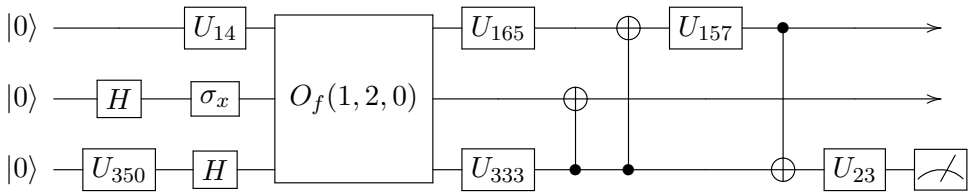

Figura 5-11: Versión simplificada del mejor programa obtenido para la función B_3, sin mediciones parciales ($p_{e_{max}} = 0.2535$). La medición final se realiza sobre el último qubit.

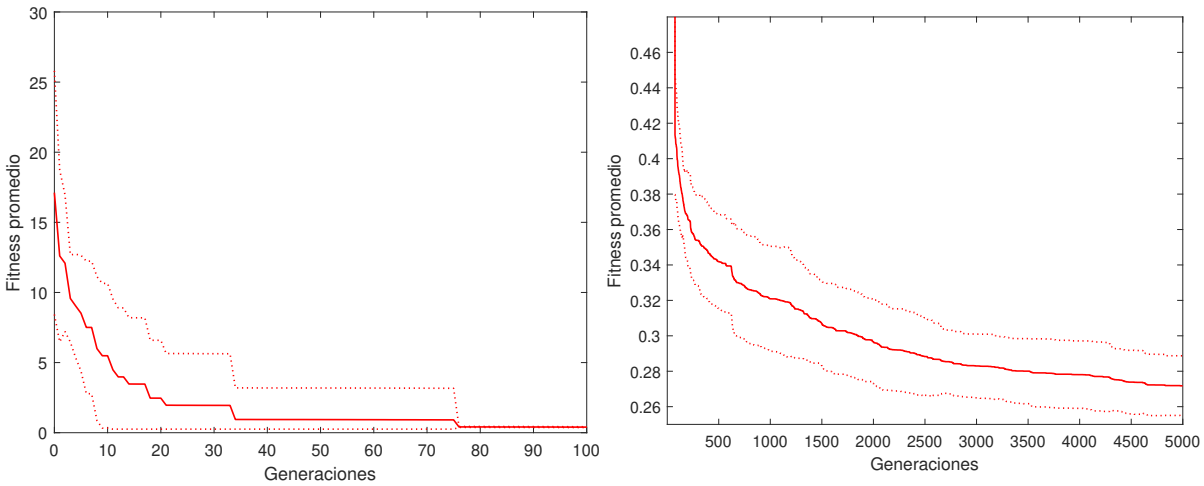

Figura 5-12: Fitness promedio de las 20 corridas a lo largo de diferentes generaciones para la función B_4, sin mediciones parciales, las líneas punteadas indican la desviación estándar.

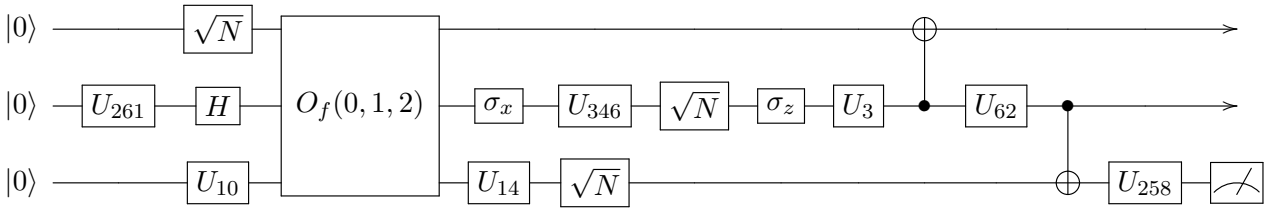

Figura 5-13: Versión simplificada del mejor programa obtenido para la función B_4, sin mediciones parciales ($p_{e_{max}} = 0.2564$). La medición final se realiza sobre el último qubit.

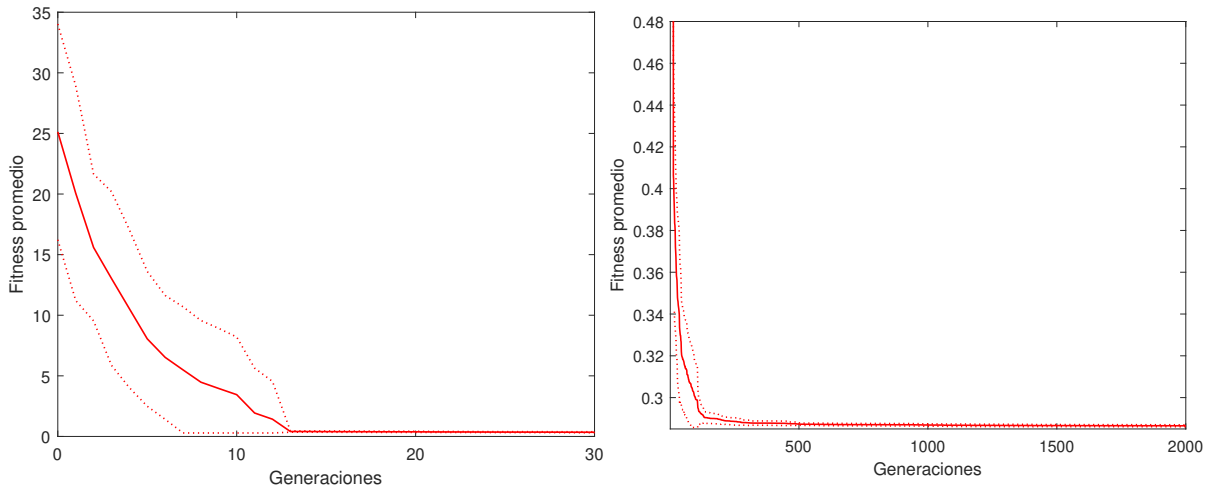

Figura 5-14: Fitness promedio de las 20 corridas a lo largo de diferentes generaciones para la función B_1, utilizando mediciones parciales, las líneas punteadas indican la desviación estándar.

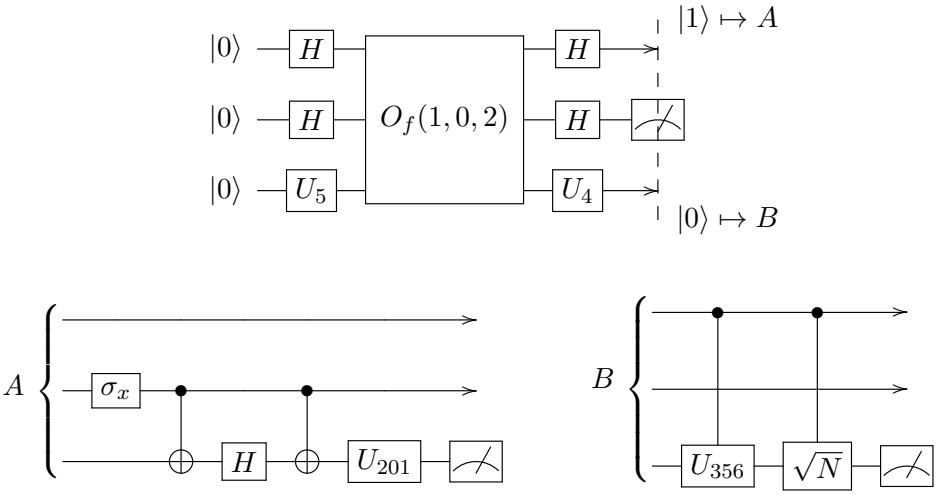

Figura 5-15: Versión simplificada del mejor programa obtenido para la función B_1, utilizando mediciones parciales ($p_{e_{max}} = 0.2859$). Las mediciones con una línea punteada atravesada representan mediciones parciales utilizadas por el programa para decidir que operaciones realizar después; se indica la siguiente parte mediante $|Resultado\rangle \mapsto Circuito$, cuando una de las dos opciones no aparece significa que la ejecución del programa termina ahí. La medición final se realiza sobre el último qubit.

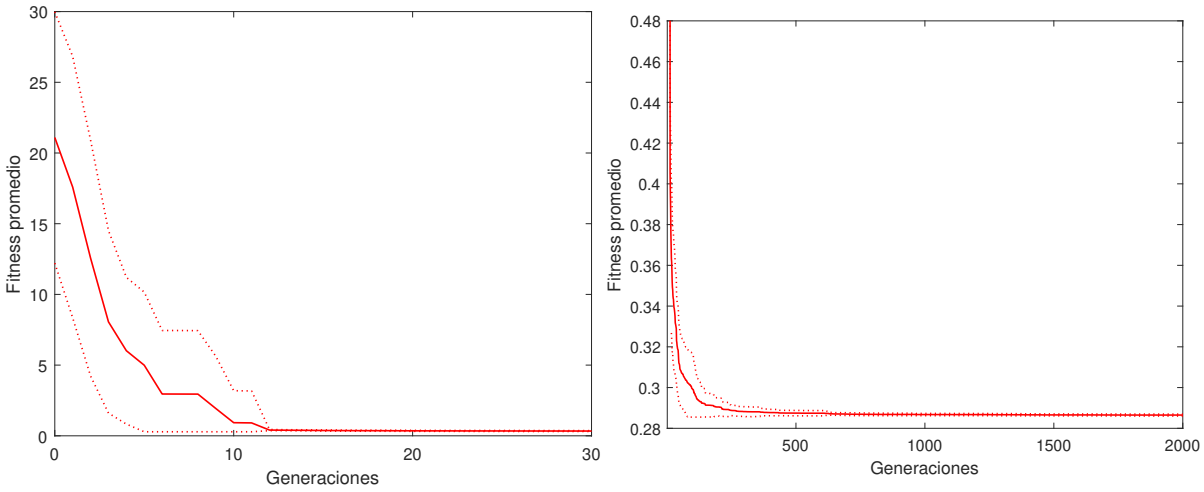

Figura 5-16: Fitness promedio de las 20 corridas a lo largo de diferentes generaciones para la función B_2, utilizando mediciones parciales, las líneas punteadas indican la desviación estándar.

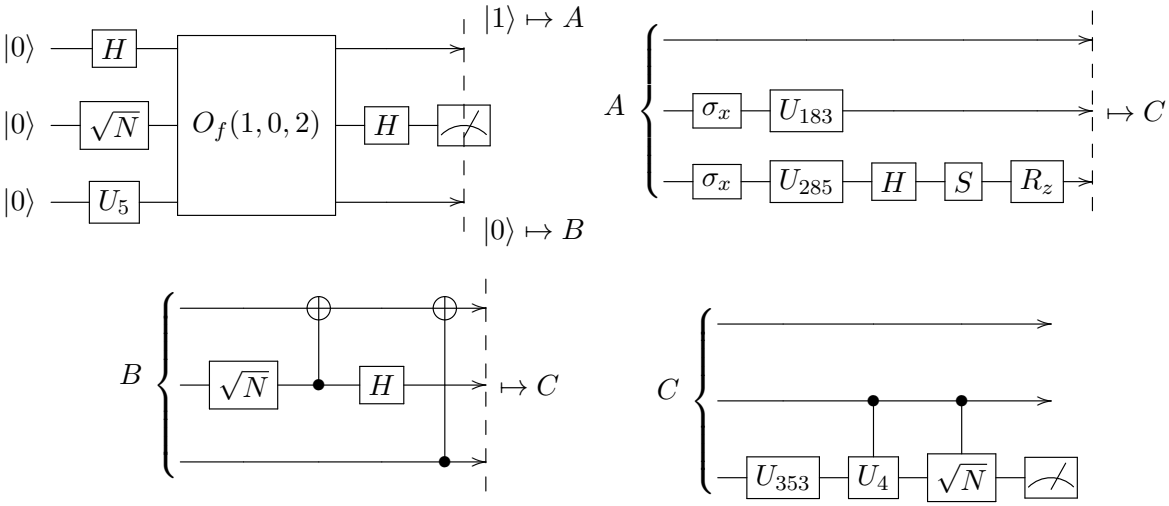

Figura 5-17: Versión simplificada del mejor programa obtenido para la función B_2, utilizando mediciones parciales ($p_{e_{max}} = 0.2857$). Las mediciones con una línea punteada atravesada representan mediciones parciales utilizadas por el programa para decidir que operaciones realizar después; se indica la siguiente parte mediante $|Resultado\rangle \mapsto Circuito$, cuando una de las dos opciones no aparece significa que la ejecución del programa termina ahí. La medición final se realiza sobre el último qubit.

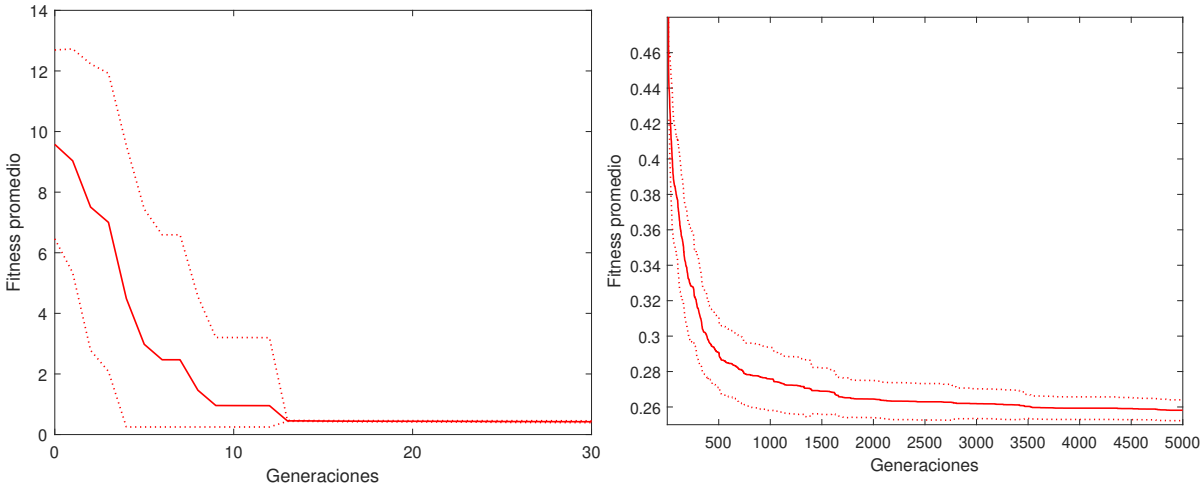

Figura 5-18: Fitness promedio de las 20 corridas a lo largo de diferentes generaciones para la función B_3, utilizando mediciones parciales, las líneas punteadas indican la desviación estándar.

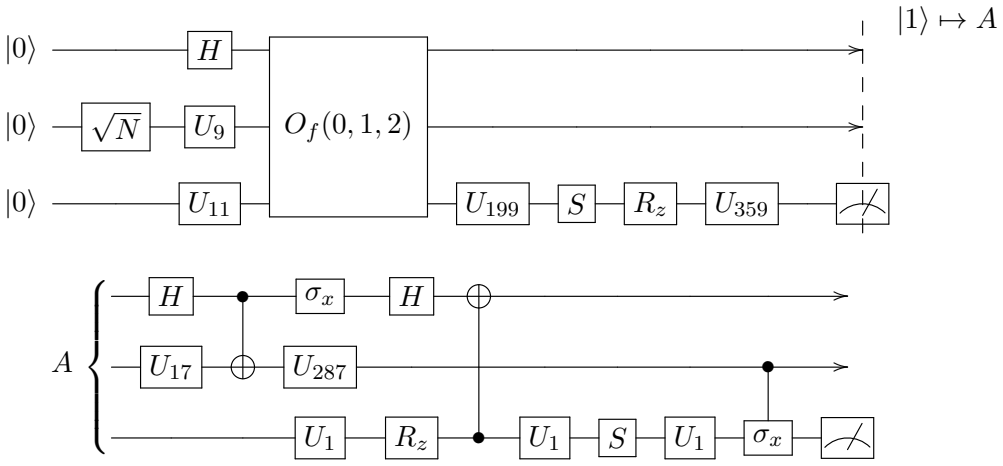

Figura 5-19: Versión simplificada del mejor programa obtenido para la función B_3, utilizando mediciones parciales ($p_{e_{max}} = 0.2527$). Las mediciones con una línea punteada atravesada representan mediciones parciales utilizadas por el programa para decidir que operaciones realizar después; se indica la siguiente parte mediante $|Resultado\rangle \mapsto Circuito$, cuando una de las dos opciones no aparece significa que la ejecución del programa termina ahí. La medición final se realiza sobre el último qubit.

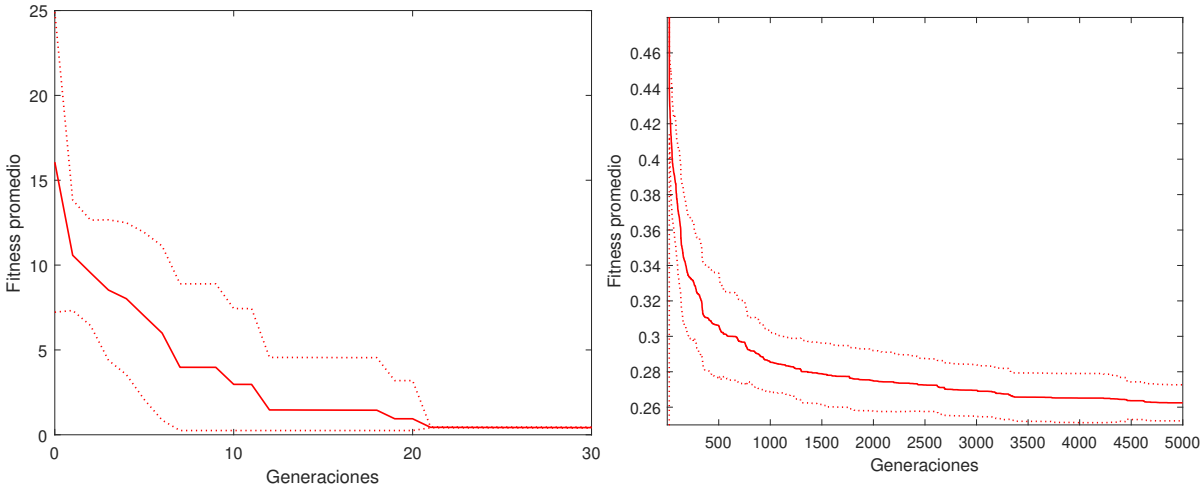

Figura 5-20: Fitness promedio de las 20 corridas a lo largo de diferentes generaciones para la función B_4, utilizando mediciones parciales, las líneas punteadas indican la desviación estándar.

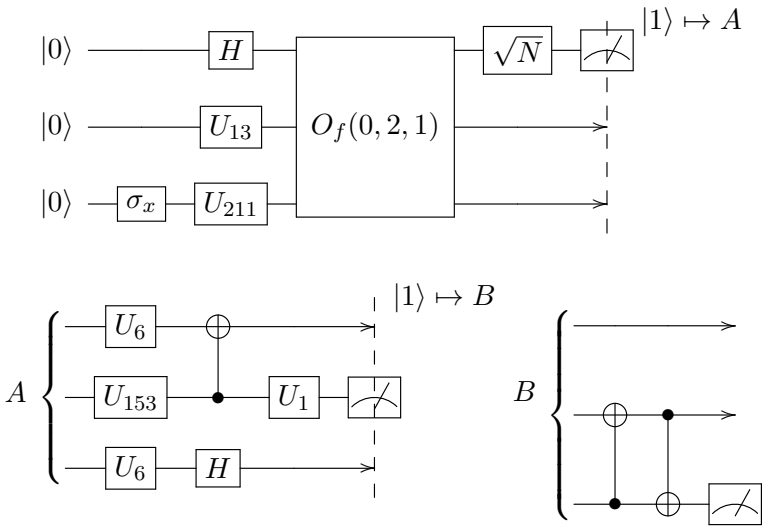

Figura 5-21: Versión simplificada del mejor programa obtenido para la función B_4, utilizando mediciones parciales ($p_{e_{max}} = 0.2536$). Las mediciones con una línea punteada atravesada representan mediciones parciales utilizadas por el programa para decidir que operaciones realizar después; se indica la siguiente parte mediante $|Resultado\rangle \mapsto Circuito$, cuando una de las dos opciones no aparece significa que la ejecución del programa termina ahí. La medición final se realiza sobre el último qubit.

En primer lugar, se puede observar que en todos los casos y para todas las corridas, el algoritmo LGP fue capaz de obtener un algoritmo con menor probabilidad de error máxima que la mínima para un algoritmo clásico. Vale la pena recordar que el fitness del individuo es equivalente a la probabilidad máxima de error si para ningún caso su probabilidad de error es mayor al umbral preestablecido.

Función	Sin medición	Sin medición (simplificado)	Con medición	Con medición (simplificado)
B_1	0.2863	0.2879	0.2857	0.2859
B_2	0.2870	0.2891	0.2857	0.2857
B_3	0.2533	0.2535	0.2522	0.2527
B_4	0.2540	0.2564	0.2533	0.2536

Tabla 5-6: Error de los mejores algoritmos obtenidos.

Al comparar los experimentos en los que se permitió utilizar mediciones parciales para alterar el flujo del algoritmo, no se observó mejoría alguna en la calidad de la mejor solución, aunque si una diferencia muy notoria en el proceso de evolución y convergencia (ver apéndice B). En los experimentos en los que no se permitió el uso de mediciones parciales se observó un proceso de evolución mucho más lento y accidentado en contraste con los experimentos en los que si se permitió; en muchos casos el resultado de la corrida no fue cercano a los mejores valores encontrados por esta técnica.

Para todos estos problemas, la elección de la respuesta (algoritmo) no fue necesariamente aquella con el mejor fitness, también se tomó en consideración la complejidad del algoritmo, ya que en algunos casos el sistema produjo respuestas con una mayor cantidad de operaciones de ramificación con apenas mejorías en el fitness. Por ejemplo, para la función B_3 encontró en la corrida 1 una respuesta con un fitness de 0.2522 y 5 operaciones de ramificación y en la corrida 17 una respuesta con un fitness de 0.2527 y 2 operaciones de ramificación, por lo que se seleccionó esta última como la respuesta para este problema.

Durante la simplificación manual de los algoritmos se eliminaron compuertas que producían correcciones mínimas al fitness. Por ejemplo, la solución original para la función B_1, que utiliza

mediciones parciales, tiene una probabilidad de error máxima de 0.2857 y la versión simplificada tiene una probabilidad de error máxima de 0.2859 debido a la eliminación de las compuertas U_1 y U_{359} de la parte A. El objetivo de estas reducciones es visualizar con mayor facilidad que es lo que está haciendo el algoritmo para resolver el problema.

En general, las funciones B_1 y B_2 parecen tener una cota mínima de error mayor a las funciones B_3 y B_4, de 0.285 para las primeras dos y de 0.25 para las últimas. La principal razón de esto puede estar relacionado a que tanto la función B_1 y B_2 son funciones casi balanceadas a diferencia de las funciones B_3 y B_4 que están mucho más cargadas hacia uno de los valores. También es interesante notar que el proceso de convergencia para las primeras dos funciones es mucho más estable que para las últimas dos; para las primeras dos funciones, en promedio, la corrida converge tras 500 generaciones mientras que para las últimas dos la corrida puede no converge, cerca del mínimo observado, en las 5000 generaciones.

Conclusión

El sistema descrito en este trabajo se mostró capaz de encontrar algoritmos viables en todos los problemas propuestos. Para el caso de identificación de oráculos balanceados y constantes encontró, esencialmente, el mismo algoritmo propuesto por Deutsch-Josza. Para el problema de búsqueda de un elemento en una base de datos no ordenada de cuatro elementos encontró un algoritmo considerablemente más compacto que el algoritmo de Grover. Para los problemas de evaluación de funciones booleanas encontró algoritmos con menor error que el mejor algoritmo probabilístico clásico posible, vale la pena mencionar que este tipo de problemas son de especial interés para el cómputo, debido a que una gran cantidad de problemas diferentes se puede pensar como un problema de evaluación de una función booleana, por lo que el estudio de las soluciones de estos problemas puede asistir en el desarrollo de mejores algoritmos cuánticos para diferentes problemas.

A pesar de la idea intuitiva de la utilización de mediciones parciales a mitad del algoritmo para dirigir el comportamiento del algoritmo, en los experimentos realizados con las funciones booleanas B_i no se observó ninguna mejoría significativa en la calidad de los algoritmos que utilizan dichas instrucciones y, aunque no se descarta la posibilidad de que puedan resultar útiles en otras situaciones, no parecen tener ninguna utilidad práctica para los algoritmos, a lo cual debe añadirse que, en general, son algoritmos de mayor complejidad por lo que los algoritmos sin mediciones parciales resultan más atractivos. La capacidad del sistema para generar algoritmos de una calidad similar en ambos casos puede deberse a dos posibles factores: el primero es que,

debido al principio de la medición diferida, el sistema puede crear algoritmos equivalentes con ambos conjuntos de operadores; por otro lado, podría ser que las mediciones parciales sean simplemente innecesarias. De cualquier manera, dichas instrucciones facilitan la convergencia del algoritmo, por lo que pueden ser utilizadas para buscar cotas de error aproximadas de manera más eficiente.

Es interesante observar que, para los problemas de funciones booleanas, el sistema encontró que era de particular utilidad poner en una superposición pesada al qubit objetivo de la función oráculo, utilizando las compuertas U_θ, previo a su aplicación. En la gran mayoría de las algoritmos observados se encontró este comportamiento, incluyendo aquellos que no se muestran explícitamente en este trabajo. Para los dos qubits que se utilizan para definir x suele utilizar superposiciones balanceadas, generadas mediante H, $\sqrt{N}$ o alguna rotación U_θ con algún ángulo que deje al sistema en una superposición casi balanceada. Un estudio más profundo de este comportamiento podría resultar de provecho para el desarrollo de mejores algoritmos cuánticos en un futuro.

Partiendo de la observación anterior, salta a la vista que diseñar herramientas para el estudio de las soluciones de manera poblacional, más que como algoritmos individuales, podría traer a la luz nuevas ideas para diseñar algoritmos cuánticos eficientes.

El sistema, en su estado actual, es capaz de obtener resultados útiles, aunque vale la pena resaltar que el algoritmo LGP no está adaptado para resolver problemas en el dominio cuántico, por lo que realizar modificaciones que puedan facilitar la exploración del espacio de algoritmos cuánticos podría mejorar, considerablemente, la calidad de los algoritmos y reducir el tiempo de convergencia de proceso. Modificaciones como dotar al sistema de la capacidad de asignar o remover operaciones de control a cualquier compuerta que ya esté presente en el programa así como la identificación de intrones en tiempo de ejecución podrían aumentar el poder de este sistema, como parece inferirse de algunas de las simplificaciones realizadas, en especial cuando se intente construir algoritmos cuánticos más grandes.

Operaciones

Operaciones utilizadas en el presente trabajo:

$$\sigma_x = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \qquad \sigma_y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}$$

$$\sigma_z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \qquad S = \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}$$

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \qquad \sqrt{N} = \frac{1}{2} \begin{bmatrix} 1+i & 1-i \\ 1-i & 1+i \end{bmatrix}$$

$$R_z = \begin{bmatrix} \cos(\pi/16) + i\sin(\pi/16) & 0 \\ 0 & \cos(\pi/16) - i\sin(\pi/16) \end{bmatrix}$$

$$U_\theta = \begin{bmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{bmatrix} \qquad CNOT = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Funciones booleanas

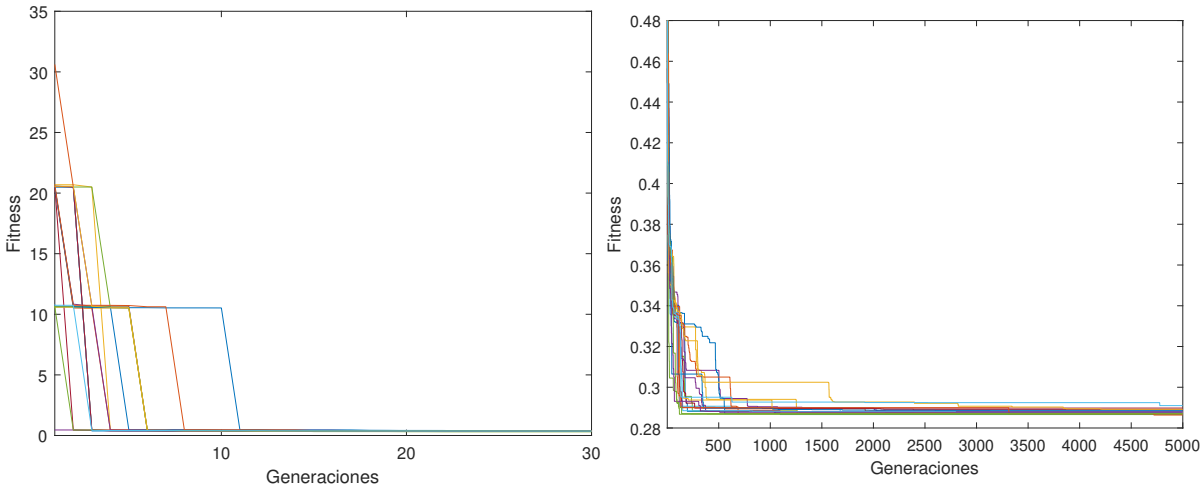

Figura B-1: Fitness de los individuos a lo largo de diferentes generaciones para la función B_1, sin mediciones parciales, para cada una de las 20 corridas.

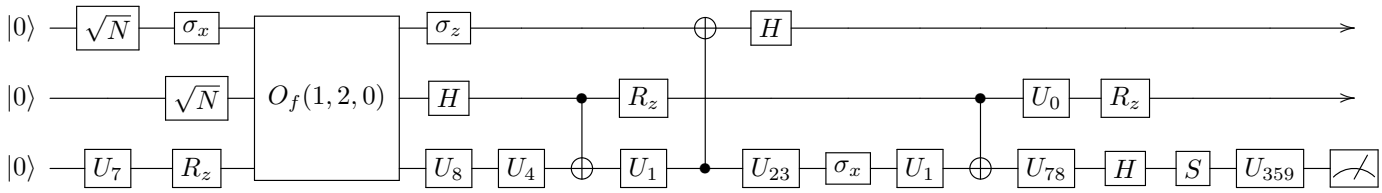

Figura B-2: Mejor solución obtenida en una de las ejecuciones del algoritmo para la función B_1 ($p_{e_{max}} = 0.2863$). La medición final se realiza sobre el último qubit.

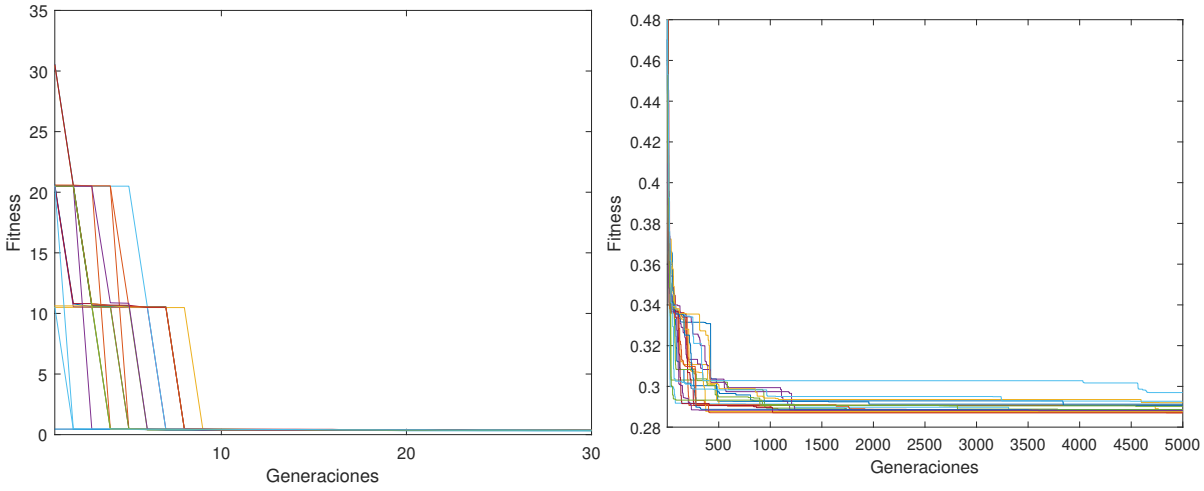

Figura B-3: Fitness de los individuos a lo largo de diferentes generaciones para la función B_2, sin mediciones parciales, para cada una de las 20 corridas.

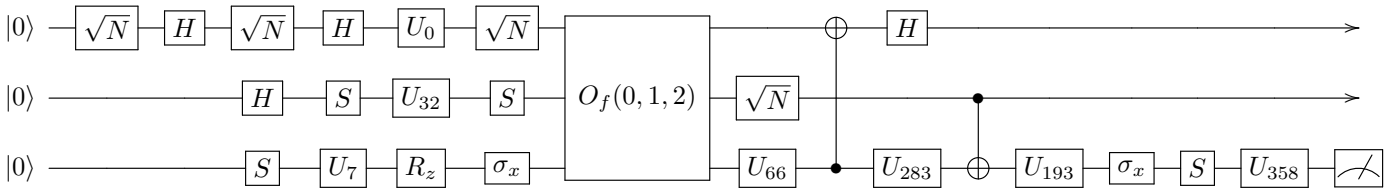

Figura B-4: Mejor solución obtenida en una de las ejecuciones del algoritmo para la función B_2 ($p_{e_{max}} = 0.2870$). La medición final se realiza sobre el último qubit.

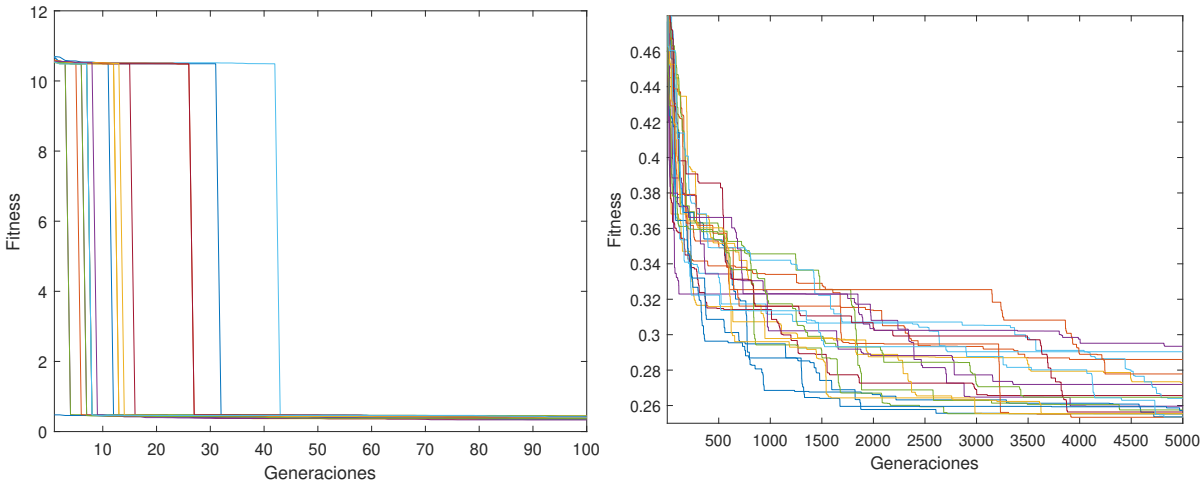

Figura B-5: Fitness de los individuos a lo largo de diferentes generaciones para la función B_3, sin mediciones parciales, para cada una de las 20 corridas.

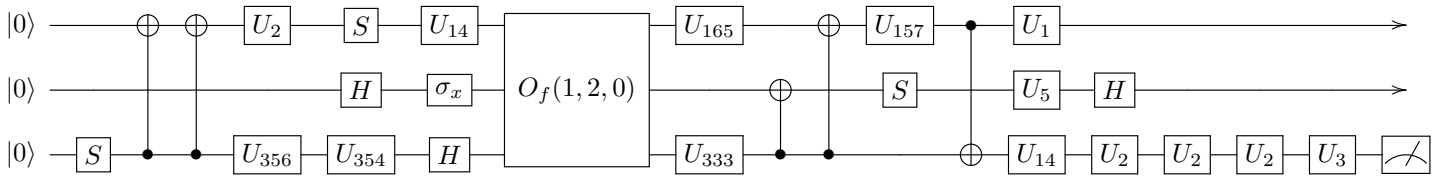

Figura B-6: Mejor solución obtenida en una de las ejecuciones del algoritmo para la función B_3 ($p_{e_{max}} = 0.2533$). La medición final se realiza sobre el último qubit.

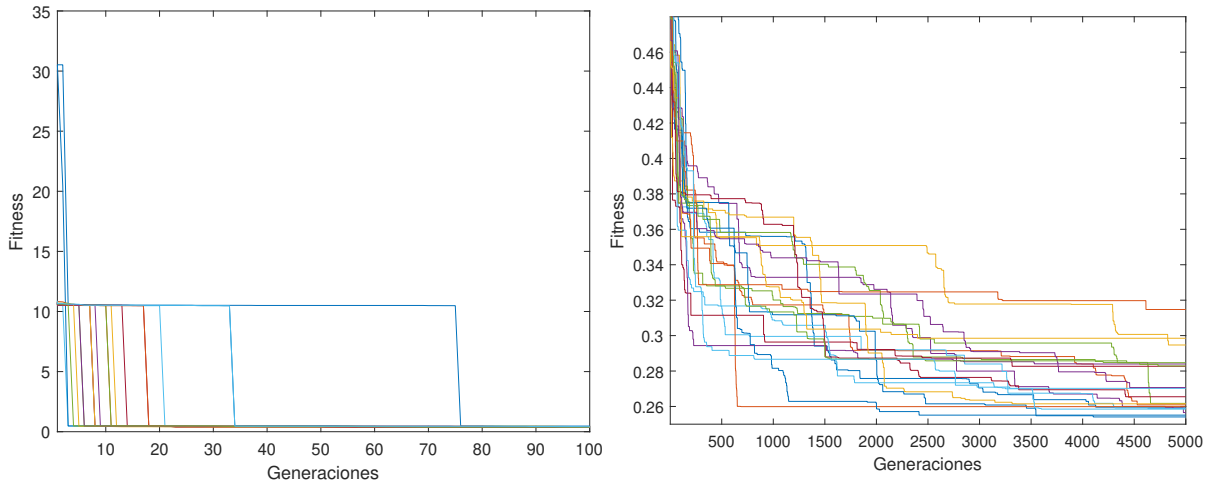

Figura B-7: Fitness de los individuos a lo largo de diferentes generaciones para la función B_4, sin mediciones parciales, para cada una de las 20 corridas.

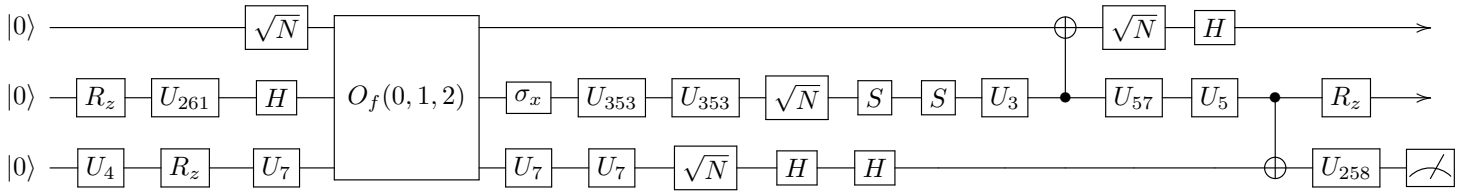

Figura B-8: Mejor solución obtenida en una de las ejecuciones del algoritmo para la función B_4 ($p_{e_{max}} = 0.2540$). La medición final se realiza sobre el último qubit.

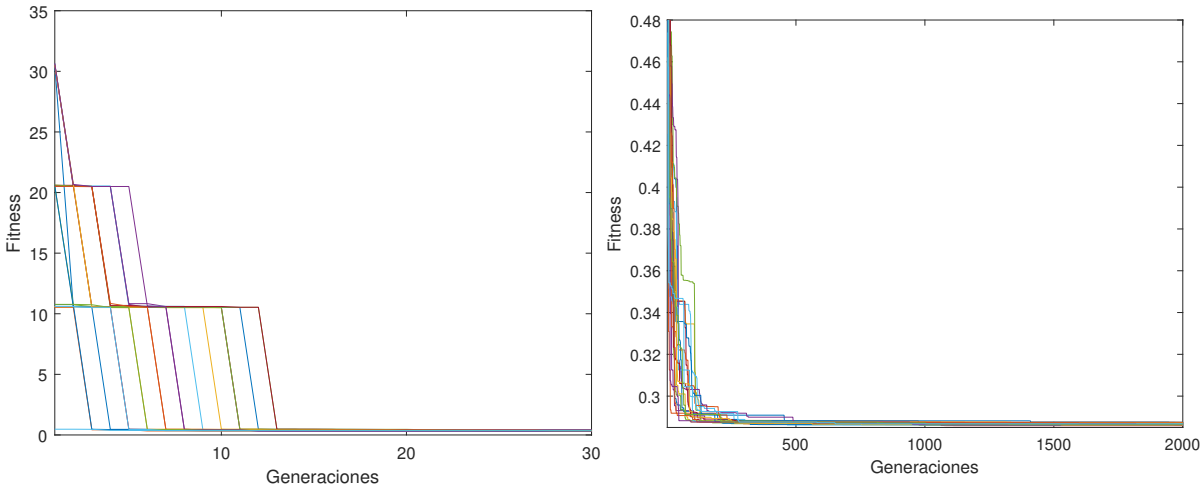

Figura B-9: Fitness de los individuos a lo largo de diferentes generaciones para la función B_1, utilizando mediciones parciales, para cada una de las 20 corridas.

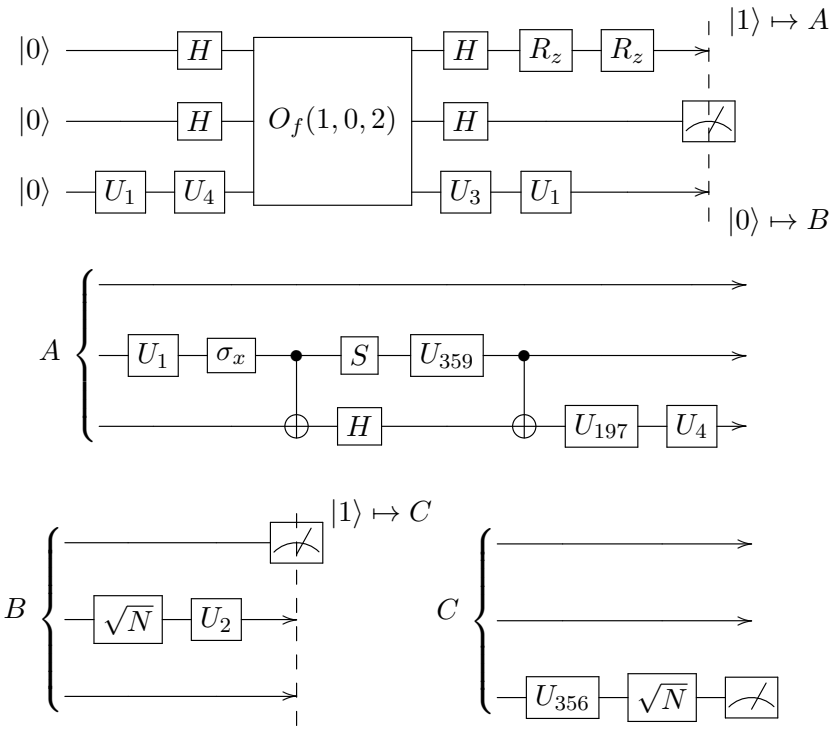

Figura B-10: Mejor solución obtenida en una de las ejecuciones del algoritmo para la función B_1 ($p_{e_{max}} = 0.2857$). Las mediciones con una línea punteada atravesada representan mediciones parciales utilizadas por el programa para decidir que operaciones realizar después; se indica la siguiente parte mediante $|Resultado\rangle \mapsto Circuito$, cuando una de las dos opciones no aparece significa que la ejecución del programa termina ahí. La medición final se realiza sobre el último qubit.

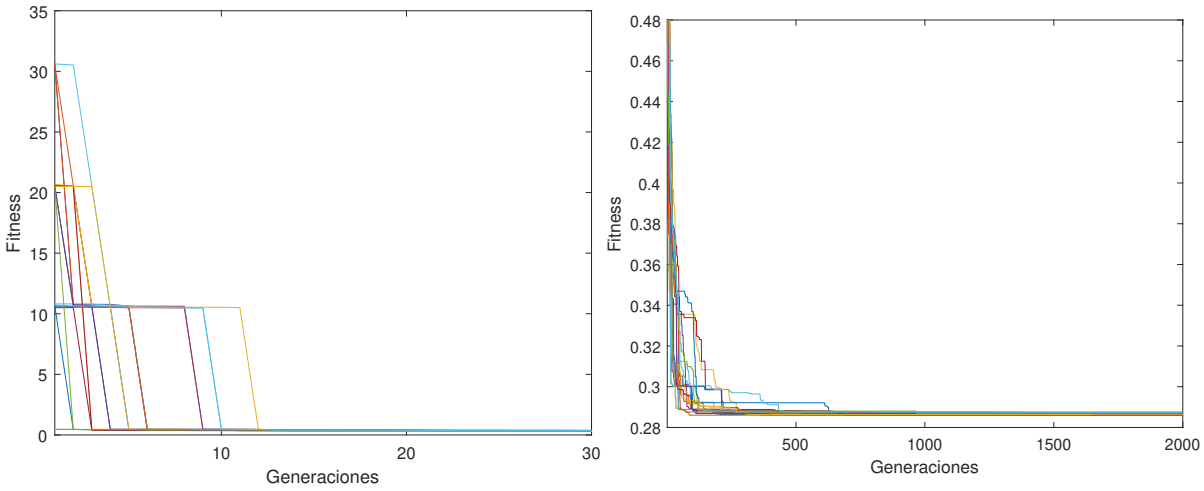

Figura B-11: Fitness de los individuos a lo largo de diferentes generaciones para la función B_2, utilizando mediciones parciales, para cada una de las 20 corridas.

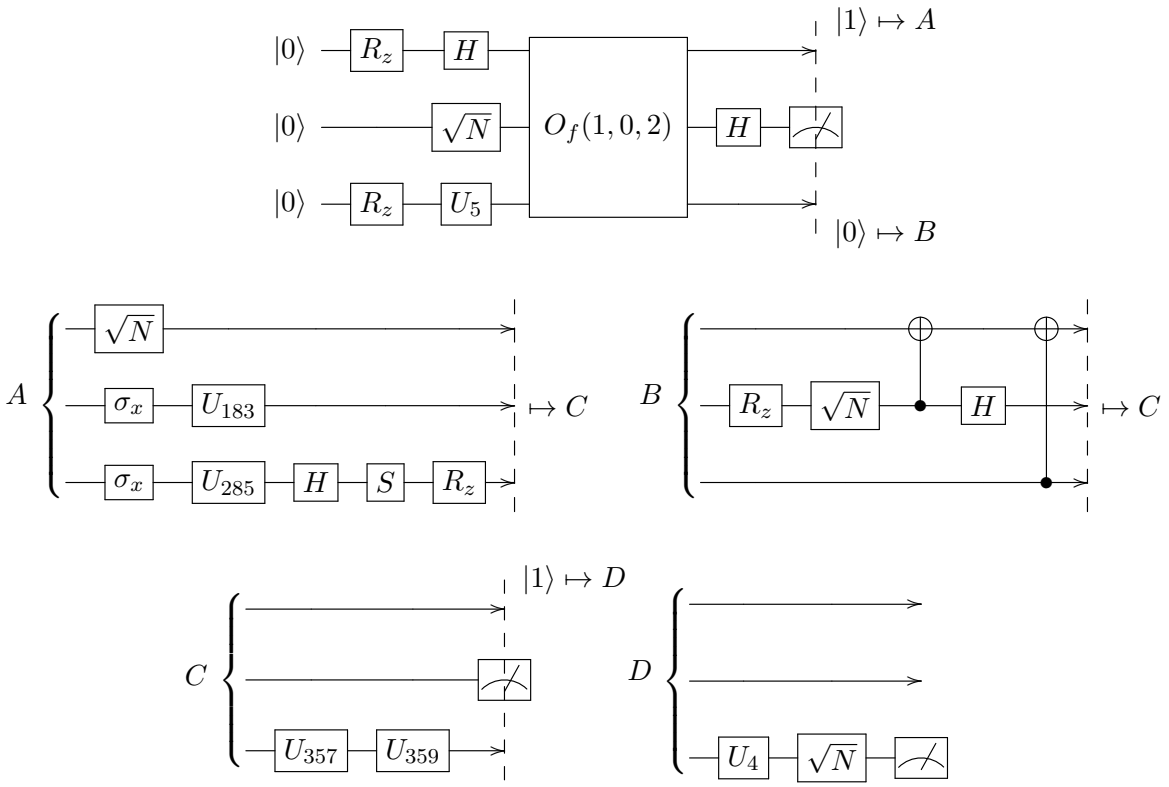

Figura B-12: Mejor solución obtenida en una de las ejecuciones del algoritmo para la función B_2 ($p_{e_{max}} = 0.2857$). Las mediciones con una línea punteada atravesada representan mediciones parciales utilizadas por el programa para decidir que operaciones realizar después; se indica la siguiente parte mediante $|Resultado\rangle \mapsto Circuito$, cuando una de las dos opciones no aparece significa que la ejecución del programa termina ahí. La medición final se realiza sobre el último qubit.

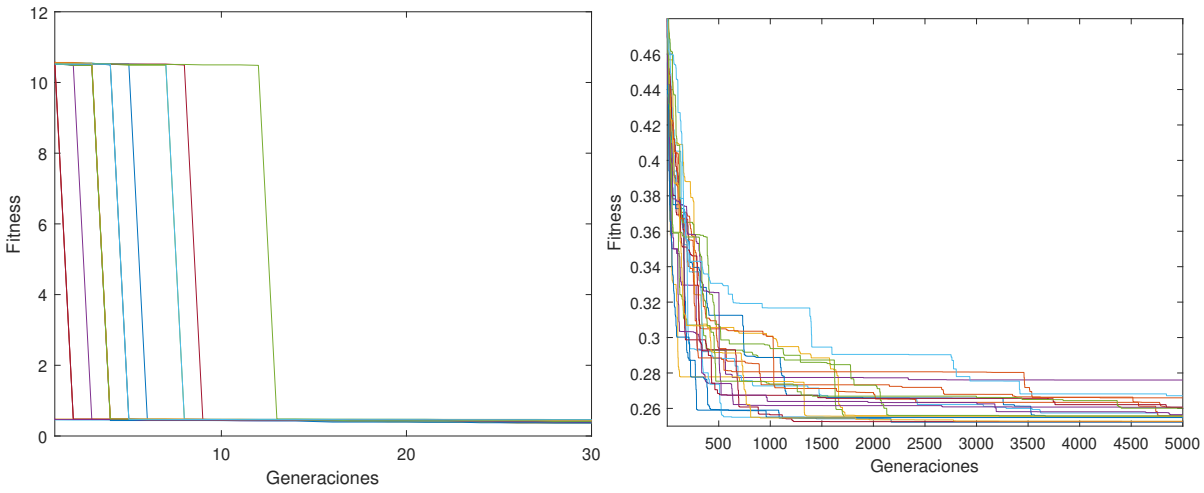

Figura B-13: Fitness de los individuos a lo largo de diferentes generaciones para la función B_3, utilizando mediciones parciales, para cada una de las 20 corridas.

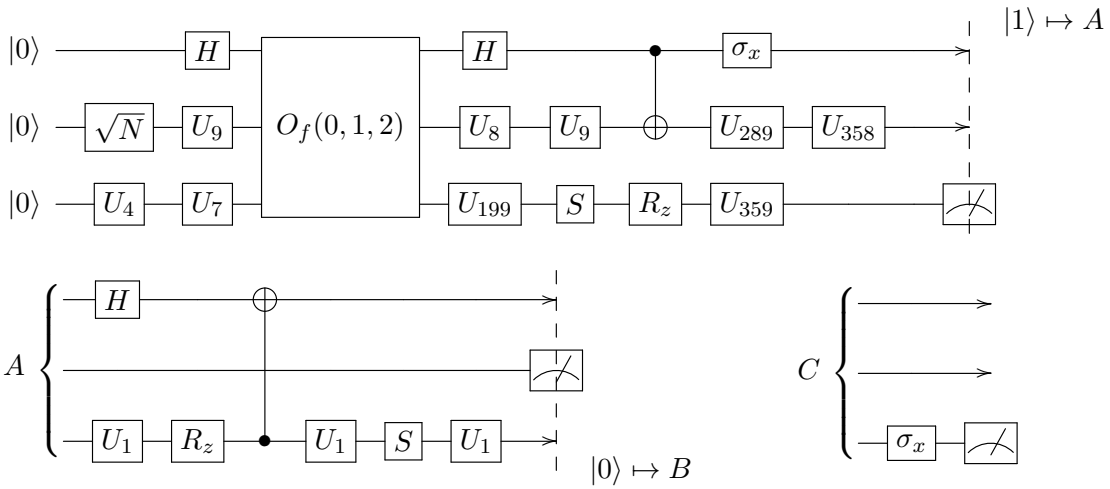

Figura B-14: Mejor solución obtenida en una de las ejecuciones del algoritmo para la función B_3 ($p_{e_{max}} = 0.2522$). Las mediciones con una línea punteada atravesada representan mediciones parciales utilizadas por el programa para decidir que operaciones realizar después; se indica la siguiente parte mediante $|Resultado\rangle \mapsto Circuito$, cuando una de las dos opciones no aparece significa que la ejecución del programa termina ahí. La medición final se realiza sobre el último qubit.

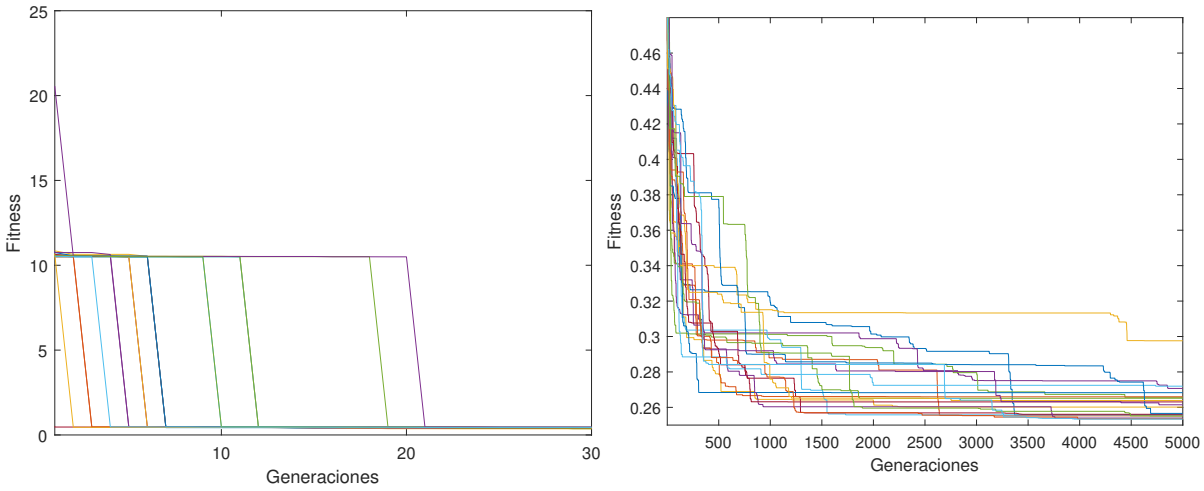

Figura B-15: Fitness de los individuos a lo largo de diferentes generaciones para la función B_4, utilizando mediciones parciales, para cada una de las 20 corridas.

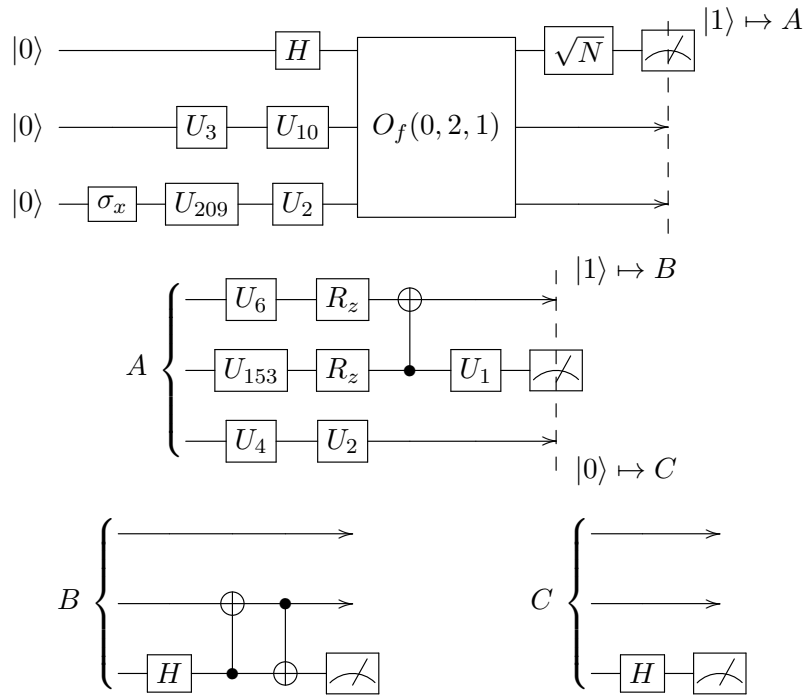

Figura B-16: Mejor solución obtenida en una de las ejecuciones del algoritmo para la función B_4 ($p_{e_{max}} = 0.2536$). Las mediciones con una línea punteada atravesada representan mediciones parciales utilizadas por el programa para decidir que operaciones realizar después; se indica la siguiente parte mediante $|Resultado\rangle \mapsto Circuito$, cuando una de las dos opciones no aparece significa que la ejecución del programa termina ahí. La medición final se realiza sobre el último qubit.

Bibliografía

[1] Andris Ambainis, Kazuo Iwama, Akinori Kawachi, Hiroyuki Masuda, Raymond H. Putra, and Shigeru Yamashita. Quantum identification of boolean oracles. *arXiv:quant-ph/0403056*, 2004.

[2] Howard Barnum, Hebert J. Bernstein, and Lee Spector. Quantum circuits for or and and of ors. *J. Phys. A: Math*, 2000.

[3] Robert Beals, Harry Buhrman, Richard Cleve, Michele Mosca, and Ronald de Wolf. Quantum lower bounds by polynomials. *arXiv:quant-ph/9802049v3*, 1998.

[4] Giuliano Benenti, Giulio Casati, and Giuliano Strini. *Principles of Quantum Computation and Information Volume I: Basic Concepts*. World Scientific, 2004.

[5] Giuliano Benenti, Giulio Casati, and Giuliano Strini. *Principles of Quantum Computation and Information Volume II: Basic Tools and Special Topics*. World Scientific, 2007.

[6] Ethan Bernstein and Umesh Vazirani. Quantum complexity theory. *SIAM Journal on Computing*, 1997.

[7] H. Buhrman, R. Cleve, R. de Wolf, and Ch. Zalka. Bounds for small-error and zero-error quantum algorithms. *arXiv:cs/9904019*, 1999.

[8] Richard Cleve. Quantum algorithms revisited. *arXiv:quant-ph/9708016*, 1997.

[9] Richard Cleve. An introduction to quantum complexity theory. *arXiv:quant-ph/9906111*, 1998.

[10] David Deutsch and Richard Jozsa. Rapid solution of problems by quantum computation. *Proc. R. Soc. Lond. A*, 1992.

[11] Lov. K. Grover. A fast quantum mechanical algorithm for database search. *Proceedings, 28th Annual ACM Symposium on the Theory of Computing*, 1996.

[12] Gregory S. Hornby. Alps: The age-layered population structure for reducing the problem of premature convergence. *GECCO'06*, 2006.

[13] John R. Koza. *Genetic Programming.* MIT Press, 1998.

[14] Chris Lomont. Quantum circuit identities. *arXiv:quant-ph/0307111*, 2003.

[15] Markus, F. Bramier, and Wolfgang Banzhaf. *Linear Genetic Programming.* Springer, 2007.

[16] Michael A. Nielsen and Issac L. Chuang. *Quantum Computation and Quantum Information.* Cambridge University Press, 2010.

[17] Riccardo Poli, Wiliam B. Langdon, and Nicholas F. McPhee. *A Field Guide to Genetic Programming.* Published via http://lulu.com and freely available at http://www.gp-field-guide.org.uk, 2008.

[18] Michael Saks and Avi Wigderson. Probabilistic boolean decision trees and the complexity of evaluating game trees. *27th Annual Symposium on Foundations of Computer Science*, 1986.

[19] Miklos Santha. On the monte carlo boolean decision tree complexity of read-once formulae. *Random Structures and Algorithms*, 1995.

[20] Daniel R. Simon. On the power of quantum computation. *SIAM Journal on Computing*, Volume 26 Issue 5, Oct. 1997 Pages 1474-1483.

[21] Lee Spector. *Automatic Quantum Computer Programming A Genetic Programming Approach.* Klvwer Academic Publishers, 2004.

[22] Lee Spector, Howard Barnum, and Herbert J. Bernstein. Genetic programming for quantum computers. *Genetic Programming 1998: Proceeding of the Third Annual Conference*, 1998.

[23] Lee Spector, Howard Barnum, and Herbert J. Bernstein. Finding a better-than-classical quantum and/or algorithm using genetic programming. *Proceedings of the 1999 Congress on Evolutionary Computation*, 1999.

[24] Lee Spector, U. M O'Reilly, and W. Langdon. *Advances in Genetic Programming, Volume 3.* MIT Press, 1999.