Universidad de Oviedo

ESCUELA POLITÉCNICA DE INGENIERÍA DE GIJÓN.

MÁSTER UNIVERSITARIO EN INGENIERÍA ÍNFORMATICA

ÁREA DE CIENCIA DE LA COMPUTACIÓN E INTELIGENCIA ARTIFICIAL

TRABAJO FIN DE MÁSTER

METAHEURÍSTICAS APLICADAS AL PROBLEMA DE LA COMPILACIÓN EN CIRCUITOS CUÁNTICOS

D. ARUFE RIVAS, Lis
TUTOR: D. Miguel Ángel González Fernández
COTUTOR: D. José Ramiro Varela Arias

FECHA: Julio 2020

Índice general

Índice de figuras

Índice de tablas

Agradecimientos

Esta investigación ha sido apoyada por el Gobierno Español en el marco de un proyecto de investigación TIN2016-79190-R y por el Principado de Asturias bajo subvención FC-GRUPIN-IDI/2018/000176.

Capítulo 1

Introducción

Esta ampliamente aceptado que la computación cuántica puede representar el próximo gran paso en el campo de la computación, ya que podría contribuir a resolver algunos problemas extremadamente difíciles. De la misma manera que las computadoras clásicas operan en puertas lógicas, la computación cuántica se basa en puertas cuánticas (qgates) que operan en bits cuánticos (qubits), cada qubit representa un estado cuántico (qstate), por ejemplo, una superposición de los dos qstates puros, denotado $|0\rangle$ y $|1\rangle$. La ejecución de un algoritmo cuántico en un hardware cuántico implica evaluar un conjunto de qgates en qubits que representan los qstates apropiados, este proceso está sujeto a algunas restricciones impuestas tanto por el algoritmo como por el hardware. Consideramos aquí la tecnología de hardware denominada procesadores Noisy Intermediate Scale Quantum (NISQ) [1], que consiste en un conjunto de qubits distribuidos en un grafo ponderado y no dirigido.

Debido al hecho de que los qgates binarios solo se pueden aplicar a qubits adyacentes, cuando se debe aplicar un qgate en dos qstates particulares, debemos asegurarnos de que estén ubicados en qubits adyacentes. Para este fin, puede ser necesario mover qstates de un qbit a otro, lo que se hace mediante puertas *swap*, es decir, puertas que solo intercambian los qstates de dos qubits adyacentes. Además, no se pueden aplicar dos puertas en el mismo qubit al mismo tiempo. Estos hechos plantean el problema de distribuir los cálculos en el hardware específico, que en la literatura se conoce como problema de compilación de circuitos cuánticos (quantum circuit compilation problem o QCCP) y puede formularse en el marco de planificación / scheduling.

Además, el hardware de computación cuántica se ve afectado por el fenómeno de la decoherencia, que degrada el rendimiento de los algoritmos cuánticos en el tiempo. Por tanto, es importante minimizar la duración de un circuito cuántico (es decir, el makespan) para reducir el efecto de la decoherencia.

En este trabajo, investigamos el desempeño de algoritmos genéticos y técnicas de búsqueda local aplicadas al QCCP. En particular, nos centraremos en el modelo descrito por Venturelli et al. en [2], ya considerado en otras propuestas [3, 4, 5]: (i) la clase de Quantum Approximate Optimization Algorithm (QAOA) aplicada al problema MaxCut [6], y (ii) una arquitectura de hardware específica inspirada en la propuesta por Rigetti Computing Inc. [7].

Aunque la compilación de circuitos cuánticos se ha considerado en la literatura durante más de una década, el Quantum Circuit Compitation Problem (QCCP) que consideramos aquí se formuló inicialmente en [2], donde los autores exploraron

el uso de planificadores temporales para resolverlo. Además, propusieron un benchmark (conocido como el conjunto de Venturelli) que se utiliza en trabajos posteriores y proporcionaron los primeros resultados para este conjunto. Estos resultados fueron superados en [3], donde los autores propusieron un procedimiento de búsqueda aleatoria voraz. Este procedimiento fue incorporado en un algoritmo genético en [4] produciendo mejoras adicionales.

En [8] se propone un método híbrido que combina Constraint Programming (CP) y Temporal Planning (TP). En este caso, los autores también consideraron dos variantes del QCCP, la primera incluye la capacidad de inicializar arbitrariamente qubits (QCCP-I), y la segunda tiene en cuenta las interacciones de diafonía entre qubits (QCCP-X). El enfoque combinado (CP + TP) superó a los anteriores en las instancias de QCCP.

En [5], los autores propusieron un procedimiento iterativo que utiliza de manera estocástica una heurística basada en rollout que explota una regla de prioridad. Este método obtiene, hasta donde sabemos, los mejores resultados hasta la fecha en las instancias QCCP de Venturelli.

También vale la pena mencionar el trabajo presentado en [9], debido al hecho de que los autores probaron que el QCCP con minimización de makespan es NP-completo.

Este trabajo está estructurado como sigue: En el capítulo 2 se explica el Quantum Approximate Optimization Algorithm aplicado al problema de MaxCut y sus fundamentos teóricos, además de dar más detalles sobre la arquitectura hardware cuántica empleada. En el capitulo 3 se formula Quantum Circuit Compilation Problem definiendo los elementos, la notación y las restricciones del problema. También se muestra un ejemplo instancia de MaxCut y una solución de esta. En el capítulo 4 se presenta el algoritmo memético basado en descomposición (DBMA), explicando como codifica y decodifica los cromosomas, lo operadores empleados, su estructura general, y la búsqueda local y heurísticos aplicados. En el capítulo 5 se detallan las estructuras de vecindad utilizadas por la búsqueda local. En el capítulo 6 se especifica el presupuesto del proyecto. En el capítulo 7 se presenta la planificación del proyecto junto con un diagrama de Gantt. En el capítulo 8 se trata el estudio experimental, donde se comparan los resultados obtenidos por el DBMA con el estado del arte. En el capítulo 9 se presentan las conclusiones, esto es las aportaciones del proyecto y el trabajo futuro previsto.

Capítulo 2

Quantum Approximate Optimization Algorithm y el problema de MaxCut

QAOA, como se propone en [6] aprovecha la computación basada en puertas cuánticas para resolver problemas de optimización combinatoria expresados como:

$$\text{maximizar: } \sum_{\alpha=1}^{m} C_\alpha(z) \tag{2.1}$$

$$z_i \in \{0,1\}\, \forall i \in \{1,\ldots,n\} \tag{2.2}$$

donde $C_\alpha(z)$ son cláusulas sobre n variables binarias de decisión z. Por lo tanto, el objetivo es encontrar la asignación de z que maximice el número de cláusulas que se satisfacen.

Para aplicar QAOA para resolver este problema, el usuario debe traducir las cláusulas $C_\alpha(z)$ en sus Hamiltonianos cuánticos equivalentes $\mathbf{C}_\alpha$, convirtiendo cada variable z_i a spin cuántico, es decir, un qubit, y luego seleccionando un número de rondas r y dos ángulos $0 \leq \beta \leq \pi$ y $0 \leq \gamma \leq 2\pi$ para cada ronda. Luego, a partir de los qubits n en un qstate

$$|+\rangle = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle, \tag{2.3}$$

el siguiente problema Hamiltoniano

$$H_C = \sum_{\alpha=1}^{m} \mathbf{C}_\alpha \tag{2.4}$$

y el Hamiltoniano correspondiente a las operaciones mix (mixing hamiltonian)

$$H_B = \sum_{j=1}^{n} \mathbf{X_j} \tag{2.5}$$

donde

$$\mathbf{X_j} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \tag{2.6}$$

es la matriz estándar de Pauli $\mathbf{X}$, se aplica alternativamente sobre las rondas r para generar el estado final

$$|\psi_r(\vec{\gamma},\vec{\beta})\rangle = \prod_{k=1}^{r} e^{-i\beta_k H_C} e^{-i\gamma_k H_B} |+\rangle^{\otimes n} \tag{2.7}$$

Este estado se mide para obtener una solución aproximada al problema definido en la ecuación (2.1), cuyo valor esperado viene dado por

$$\langle\psi_r(\vec{\gamma},\vec{\beta})|H_C|\psi_r(\vec{\gamma},\vec{\beta})\rangle \tag{2.8}$$

Si los valores de r, $\vec{\beta}$ y $\vec{\gamma}$ están bien seleccionados, el estado de los qubits después de esta transformación representará una buena solución al problema definido en la ecuación (2.1) con alta probabilidad, y la calidad de la solución aumenta con el número de rondas. La selección de $\vec{\beta}$ no es un tema trivial y hay algunas propuestas en la literatura [6]. Un enfoque habitual es comenzar desde algunos valores iniciales y luego realizar una optimización basada en simplex o gradiente.

El problema de MaxCut es un paradigma en la optimización combinatoria que resulta particularmente ventajoso para QAOA por dos razones: (i) todas las cláusulas en la función objetivo tienen la misma estructura, por lo que solo se necesita diseñar un Hamiltoniano cuántico, y (ii) las cláusulas solo implican dos variables de decisión.

En el problema de MaxCut, se nos da un grafo no dirigido $G = (V, E)$, donde $V = \{1, \ldots, n\}$ es un conjunto de nodos y E es el conjunto de arcos. El objetivo es establecer una partición del conjunto V en dos subconjuntos V_{+1} y V_{-1} de modo que se maximice el número de arcos en E conectando nodos de los dos subconjuntos, en otras palabras, el objetivo es

$$\text{maximizar:} \sum_{(i,j)\in E} \frac{1}{2}(1 - \sigma_i\sigma_j) \tag{2.9}$$

$$\sigma_k = \begin{cases} -1 & \text{si } k \in V_{-1} \\ 1 & \text{si } k \in V_{+1} \end{cases} \tag{2.10}$$

Téngase en cuenta que una sola transformación $z = (\sigma + 1)/2$ convierte las variables del espacio $\sigma \in \{-1, +1\}$ al espacio $z \in \{0, 1\}$.

En este caso, tenemos un Hamiltoniano $\mathbf{C}_\alpha$ para cada arco (i, j), que depende de estas dos variables, por lo que se puede denotar como $\mathbf{C_{i,j}}$, y dado que opera en un estado de 2 qubits, tiene un tamaño de 4×4. De la ec. (2.9), la base computacional $\{|00\rangle, |01\rangle, |10\rangle, |11\rangle\}$ puede considerarse como qstates de un sistema físico con energías 0,1,1,0 respectivamente, por lo que el Hamiltoniano está representado por la matriz

$$\mathbf{C_{i,j}} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \tag{2.11}$$

que se puede escribir como

$$\mathbf{C_{i,j}} = \frac{1}{2}(\mathbf{I} - \mathbf{Z}_i \otimes \mathbf{Z}_j) \tag{2.12}$$

donde $\mathbf{Z_i}$ y $\mathbf{Z_j}$ son la matriz de Pauli

$$\mathbf{Z} = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \tag{2.13}$$

Por lo tanto, los términos exponenciales en la ecuación (2.7) pueden expandirse considerando que

$$e^{-i\beta_k \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}} = \begin{pmatrix} \cos(\beta_k) & -i\sin(\beta_k) \\ -i\sin(\beta_k) & \cos(\beta_k) \end{pmatrix} \tag{2.14}$$

y

$$e^{-i\frac{\gamma_k}{2} \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & e^{-i\frac{\gamma_k}{2}} & 0 & 0 \\ 0 & 0 & e^{-i\frac{\gamma_k}{2}} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2.15}$$

Los operadores en las ecuaciones (2.14) y (2.15) se tienen que implementar por medio del conjunto de qgates disponibles en el computador cuántico objetivo. En particular, en las arquitecturas Rigetti[1]. El primero podría implementarse como RX($2\beta_k$) y el segundo mediante la composición de las dos puertas de 2 qubits CPHASE02(-γ_k/2) CPHASE01(-γ_k/2). Como mencionamos, para ejecutar una qgate binaria en dos qstates, estos qstates deben estar situados en qubits adyacentes. Para este propósito, es generalmente necesario el uso de una serie de qgates *swap*; una puerta *swap* opera en dos qubits intercambiando sus qstates implementando el siguiente operador

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \tag{2.16}$$

La figura 2.1 muestra los cuatro diseños de chips cuánticos caracterizados por un número creciente de qubits ($N = 4, 8, 21, 40$) propuestos por Rigetti Computing Inc. Cada qubit está identificado por un numero entero y está localizado en un nodo. Dos qubits conectados por una arista son adyacentes, y cada arista representa que una puerta de 2 qubits (*p-s* o *swap*) puede ser ejecutada en esos qubits. Las puertas *p-s* ejecutadas en aristas continuas tienen una duración $\tau_{p-s} = 3$, mientras que las puertas *p-s* ejecutadas en aristas discontinuas tienen una duración $\tau_{p-s} = 4$; las puertas *swap* tienen una duración $\tau_{swap} = 2$ y las puertas mix tienen una duración $\tau_{mix} = 1$.

[1] ver http://docs.rigetti.com/en/stable/apidocs/gates.html

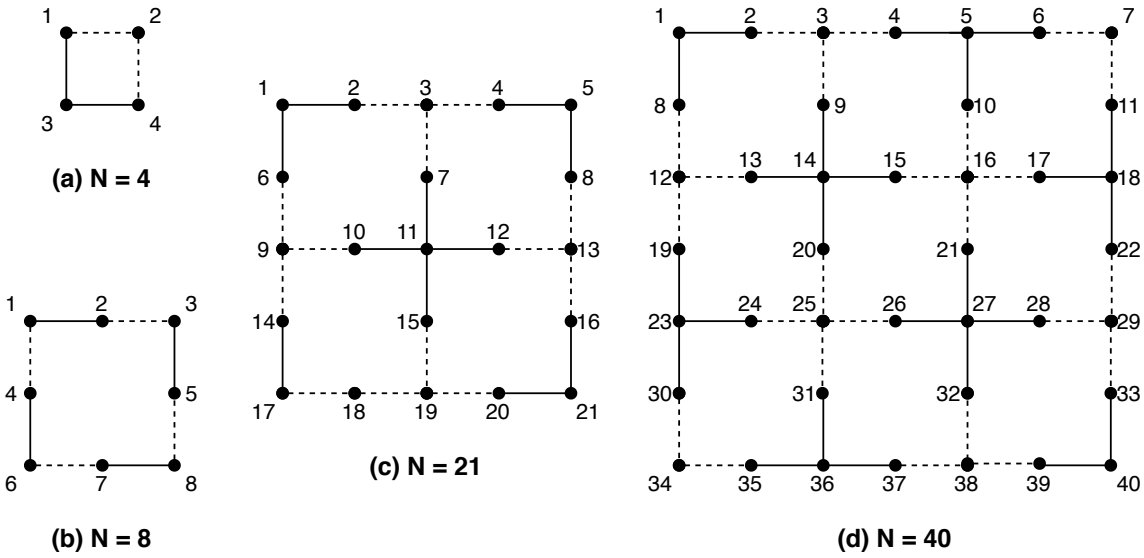


Figura 2.1: Cuatro diseños de chips cuánticos con diferente número de qubits.

Capítulo 3

Formulación de Quantum Circuit Compilation Problem

En el QCCP, viene dada una tupla $P = \langle C_0, L_0, QM \rangle$. C_0 es el circuito cuántico de entrada, que representa el algoritmo cuántico para resolver una instancia del problema de MaxCut en la arquitectura cuántica objetivo. L_0 es la asignación inicial de qstates a qubits y QM es una representación del hardware cuántico como un multígrafo.

El circuito cuántico de entrada se puede definir como $C_0 = \langle Q, P\text{-}S, MIX, \{g_{start}, g_{end}\}, TC_0 \rangle$, donde $Q = \{q_1, \ldots, q_N\}$ es el conjunto de qstates que, desde la perspectiva de la planificación y el scheduling, representan los recursos necesarios para la ejecución de cada puerta. $P\text{-}S$ y MIX son el conjunto de puertas $p\text{-}s$ y mix, que corresponden con los operadores definidos en las ecuaciones (2.15) y (2.14). Utilizamos la notación $p\text{-}s(q_i, q_j)$ y $mix(q_i)$ ya que operan respectivamente en dos y un qstates. También son consideradas dos puertas ficticias g_{start} y g_{end} que no operan en ningún qstate.

Cada puerta cuántica requiere el uso ininterrumpido de los qstates involucrados durante todo su tiempo de procesamiento, y cada qstate q_i solo puede ser procesado por una puerta al mismo tiempo.

TC_0 es un conjunto de restricciones de precedencia impuestas en los conjuntos $P\text{-}S$, MIX y $\{g_{start}, g_{end}\}$. Cada puerta en $P\text{-}S$ y MIX debe ejecutarse después de g_{start} y antes de g_{end}. De acuerdo con el número de rondas r del algoritmo QAOA, los conjuntos PS y MIX se organizan en r pasos que se deben intercalar de este modo, $P\text{-}S_1$, MIX_1, $P\text{-}S_2$, MIX_2, ..., $P\text{-}S_r$, MIX_r. Todas las puertas que pertenecen al paso $P\text{-}S_k(MIX_k)$ que implican un qstate específico q_i deben ejecutarse antes que todas las puertas que pertenecen al siguiente paso $MIX_{k+1}(P\text{-}S_{k+1})$, $k = 1, 2, \ldots, r-1$. Por lo tanto, r es el número de pasadas de compilación. Dos puertas $p\text{-}s$ que comparten un qstate son conmutativas, por lo que pueden ejecutarse en cualquier orden.

QM es una representación del hardware cuántico como un grafo no dirigido $QM = \langle V_N, E_{p-s}, E_{swap}, \tau_{mix}, \tau_{p-s}, \tau_{swap} \rangle$, $V_N = \{n_1 \ldots n_N\}$ es el conjunto de qubits (nodos) del hardware. E_{p-s} y E_{swap} son los conjuntos de aristas no dirigidas (n_k, n_l) que representan ubicaciones adyacentes donde los qstates q_i y q_j de las puertas $p\text{-}s(q_i, q_j)$ o $swap(q_i, q_j)$ pueden ser asignadas [1]; las puertas mix se pueden ejecutar en cualquier nodo. τ_{mix}, $\tau_{p\text{-}s}$ y τ_{swap} representan las duraciones de los diferentes tipos de puertas. En nuestro estudio, $\tau_{mix} = 1$, $\tau_{swap} = 2$ y τ_{p-s} es 3 si la puerta se ejecuta

[1] En este estudio, ambos conjuntos son de hecho iguales

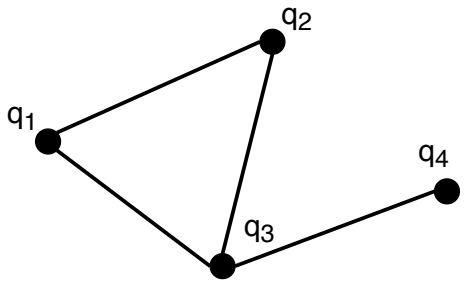
en una arista continua y es 4 cuando se ejecuta en una arista discontinua (ver Figura 2.1).

Una solución factible es una tupla $S = \langle SWAP, TC \rangle$, que extiende el circuito inicial C_0 con un conjunto $SWAP$ de puertas *swap*, agregado para garantizar las restricciones de adyacencia para las puertas en *P-S*, y un conjunto TC de restricciones de precedencia adicionales, de modo que para cada qstate q_i se impone un orden total entre el conjunto de operaciones que requieren q_i. Además, todas las puertas *p-s* y *swap* deben asignarse en qubits adyacentes en QM. El makespan de una solución corresponde al tiempo máximo de finalización de las operaciones de puerta en S.

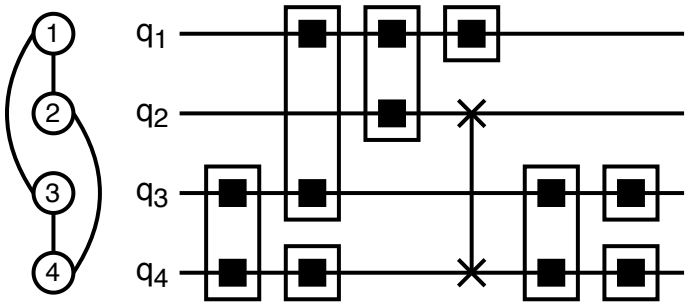
Un schedule puede verse como un grafo solución $G_S = (V_S, E_S)$, donde $V_S = \{g_{start}, g_{end}\} \cup P\text{-}S \cup SWAP$ y E_S representan el orden parcial de las operaciones en V_S definido en TC_0 y TC. La figura 3.1(c) muestra un ejemplo, en algunos casos hay dos arcos equivalentes entre dos nodos, esto es solo para enfatizar que corresponden a puertas binarias que comparten los dos qstates. Los costos en los arcos son las duraciones de las puertas de salida. El makespan del programa viene dado por el costo del camino más largo de g_{start} a g_end que se llama *camino crítico*.

El objetivo del QCCP es encontrar una solución factible con el mínimo makespan. En [9] se demuestra que QCCP es un problema NP-completo.

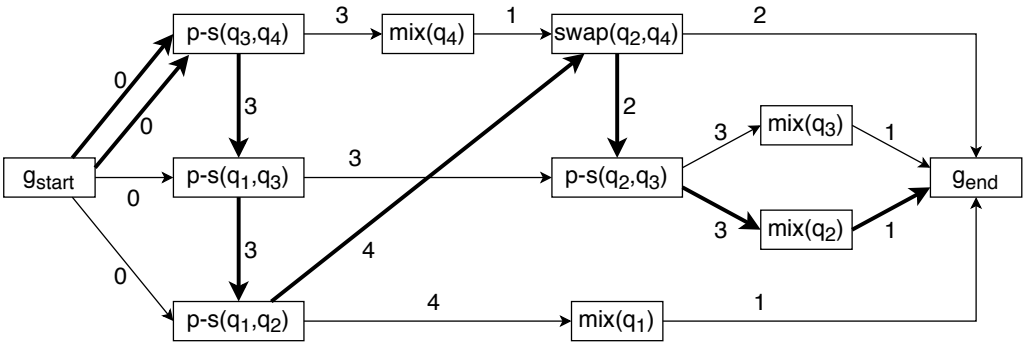
Como ejemplo, considerando una instancia de juguete en el chip con $N = 4$ (vease la Figura 2.1(a)) y solo una ronda, en la que se deben ejecutar las siguientes cuatro puertas: *p-s*(q_1, q_2), *p-s*(q_1, q_3), *p-s*(q_2, q_3) y *p-s*(q_3, q_4). La figura 3.1(a) muestra un grafo de una instancia de MaxCut con 4 nodos (el número de nodos debe ser menor o igual que el número de qubits en el chip cuántico). La figura 3.1(b) muestra una solución en forma de circuito cuántico compilado (téngase en cuenta que se necesita una puerta *swap* para garantizar el cumplimiento de adyacencia). La figura 3.1(c) muestra el grafo solución equivalente, cuyo camino crítico (que se define como el camino más largo entre el nodo g_{start} y el nodo g_end) tiene un coste de 16. Observe que las puertas *ps* y *swap* tienen dos nodos predecesores y dos nodos sucesores cada uno (a menos que operen en los mismos qstates de las puertas predecesoras o sucesoras, que se representan mediante un arco doble, como se mencionó), que corresponden a las puertas anteriores o posteriores (respectivamente) ejecutadas en los dos qstates correspondientes. Las puertas *mix* tienen solo un predecesor y un sucesor, ya que operan en un solo qstate. En cualquier caso, si una puerta es la primera (o última) que se ejecuta en un qstate, entonces su predecesor (o sucesor) es el nodo g_{start} (o nodo g_{end}). Finalmente, la Figura 3.1(d) muestra el diagrama de Gantt de la solución con makespan 16.



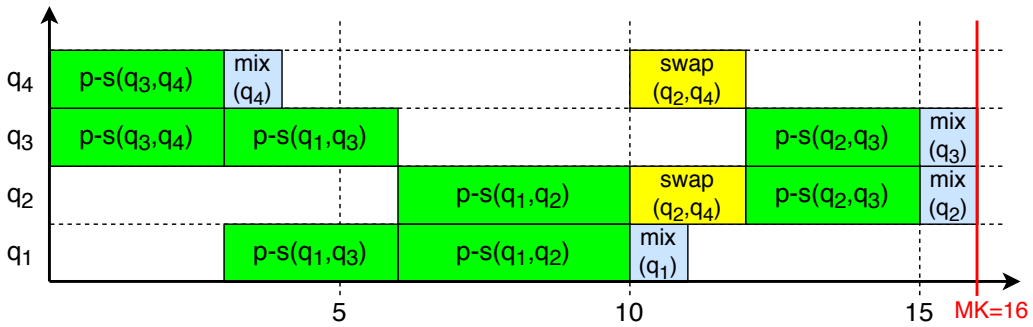
(a) Instancia de MaxCut.



(b) Circuito cuántico para la instancia de MaxCut en la Figura 3.1(a). Los rectángulos representan puertas *p-s*, los cuadrados representan puertas *mix* y las líneas simples representan puertas *swap*. Solo se incluye una ronda e inicialmente cada qstate q_i está en el qubit i. El hardware cuántico está representado a la izquierda. Cada línea horizontal representa las operaciones en un qubit a lo largo del tiempo.



(c) Grafo solución. Los arcos en negrita indican el camino crítico, cuya longitud (16) corresponde con el makespan. El nodo g_{init} representa la situación inicial, por ejemplo, el qstate q_i en el qubit i. Los arcos dobles significan que una puerta binaria toma las dos entradas del mismo recurso (el nodo g_{init} o cualquier otro no binario)



(d) Diagrama de Gantt.

Figura 3.1: Ejemplo de instancia MaxCut (a) y una solución representada por el circuito cuántico (b), un grafo solución (c) y un diagrama de Gantt (d).

Capítulo 4

El algoritmo memético basado en descomposición

Por todo lo anterior, está claro que QCCP es un problema de scheduling extremadamente complejo. En este trabajo, investigamos el uso de algoritmos meméticos; ya que se aplicaron con éxito a una variedad de problemas de scheduling [10, 11, 12], es nuestra hipótesis que también pueden ser efectivos para resolver el QCCP.

Como es habitual, el algoritmo memético propuesto combina un algoritmo genético (AG) con una búsqueda local (BL). Dada la particular estructura del problema, específicamente el hecho de que una instancia del problema está compuesta por un conjunto de r rondas y que en cada ronda se debe planificar el mismo conjunto de puertas p-s y *mix*, se propone un procedimiento incremental tal que en cada paso $p \in \{1, \ldots, r\}$ se resuelve el subproblema definido por las rondas $1, \ldots, p$. Para ser más precisos, el AG propuesto esta basado en descomposición e intenta optimizar el scheduling de las puertas p-s en la ronda p, dado un numero de soluciones al subproblema definido por rondas $1, \ldots, p-1$ (las puertas *mix* en una ronda p son planificadas de forma trivial justo después de que todas las puertas p-s que involucran al correspondiente qstate hayan sido incluidas). Este es un tipo de estrategia de optimo local que es sabido que no siempre dirige a una solución cercana a la óptima. Por lo tanto, trataremos de superar este inconveniente en la fase de BL, tratando el grafo solución completo construido hasta el subproblema $1, \ldots, p$. En las siguientes subsecciones se describirán los componentes principales del algoritmo propuesto, que se llama Decomposition-Based Memetic Algorithm (DBMA).

4.1. El esquema de codificación

Una ejecución del AG empieza a partir de un conjunto de soluciones al subproblema $1, \ldots, p-1$ calculado en pasos previos, y construye schedules para el subproblema $1, \ldots, p$. Por tanto, un cromosoma tiene que incluir una solución al subproblema $1, \ldots, p-1$ y la codificación de un schedule candidato para las puertas p-s en la ronda p. Para este fin, proponemos codificar un cromosoma mediante una tupla de la forma $(sg_{p-1}, ch1, ch2)$, donde sg_{p-1} es un grafo solución para el subproblema $1, \ldots, p-1$ y $ch1$ y $ch2$ son dos cadenas de símbolos; $ch1$ es una permutación del conjunto de puertas p-s en la ronda p y $ch2$ es una secuencia de pares de qbits adyacentes de la misma longitud que $ch1$. Esta codificación debe entenderse como que la puerta p-$s(q_i, q_j)$ en una posición de $ch1$ debe ejecutarse en el par de qubits

$\{n_k, n_l\}$ en la misma posición en $ch2$; y el schedule debe construirse teniendo en cuenta los primeros tiempos de inicio de las puertas cuánticas en los qubits después de la solución parcial del subproblema $1, \ldots, p-1$, como veremos en la Sección 4.3.

4.2. Recombinación, mutación y diversificación

Con respecto al cruce y la mutación, se puede adaptar cualquier operador ideado para la codificación de permutaciones. En este caso, el cruce puede operar en la permutación de qgates, es decir, los componentes de $ch1$ son emparejados y los pares de qubits asociados acompañan los símbolos de los hijos. Además, cada descendiente hereda el grafo solución del subproblema $1, \ldots, p-1$ de un padre. En nuestros experimentos, extendemos el conocido partial mapping crossover (PMX). Esta operación se ilustra en la Figura 4.1. Además, consideramos una mutación simple ($mut1$) que consiste en intercambiar dos posiciones (la puerta y el par de qubits) elegidas aleatoriamente.

Con los operadores anteriores, únicamente consideraríamos en el proceso evolutivo los pares de qubits localizados en las puertas p-s de la población inicial. Así que se debería considerar algún otro operador que introduzca nuevo material genético en forma de nuevos pares de qubits candidatos para las puertas p-s. Para este fin, proponemos otra mutación ($mut2$) que cambia el par de qubits asociado a p-s de forma aleatoria. Además, consideramos una mutación específica del problema como variación de $mut2$ que consiste en seleccionar uno de los pares de qubits adyacentes al actual; que es llamada $mut3$. Claramente, $mut3$ tiene un espacio de búsqueda más reducido que $mut2$, pero al mismo tiempo es razonable esperar que $mut3$ tenga más posibilidades de producir mejoras que $mut2$. Finalmente, utilizamos un operador de mutación compuesto ($mut4$) que realiza sobre un mismo cromosoma entre una y un número máximo n de mutaciones de tipo $mut1$ o $mut3$ aleatoriamente.

Como se ha mencionado anteriormente, la diversidad de pares de qubits presentes en la población depende en gran medida de los pares de qubits localizados en las puertas p-s de la población inicial. Esto ya se ha solventado en parte con los operadores de mutación $mut4$, pero las modificaciones realizadas por esta mutación solo permanecen en la población si han producido alguna mejora en el makespan. Por tanto, hay una mayor posibilidad de convergencia prematura. Para impedir esto se plantea un operador de diversificación. Este consiste en realizar, cada cierto número de generaciones sin mejora, una mutación $mut4$ a cada cromosoma que tenga un makespan repetido en la población, dejando solo uno sin alterar.

4.3. El algoritmo de decodificación

Construir un schedule a partir del cromosoma es la cuestión más crítica dadas las particulares características del QCCP. Para este fin, el algoritmo de decodificación itera sobre el componente $ch1$ del cromosoma y toma las acciones requeridas para planificar la puerta p-$s(q_i, q_j)$ en cada posición del par de qubits $\{n_k, n_l\}$ en la misma posición de $ch2$. Estas acciones son introducir las puertas *swap* necesarias para mover los qstates q_i y q_j a los qubits n_k y n_l. En general, puede haber muchas posibilidades para hacerlo y no está claro cuál es el más apropiada en cada circunstancia. Además, no es fácil codificar cada una entre el conjunto completo de opciones

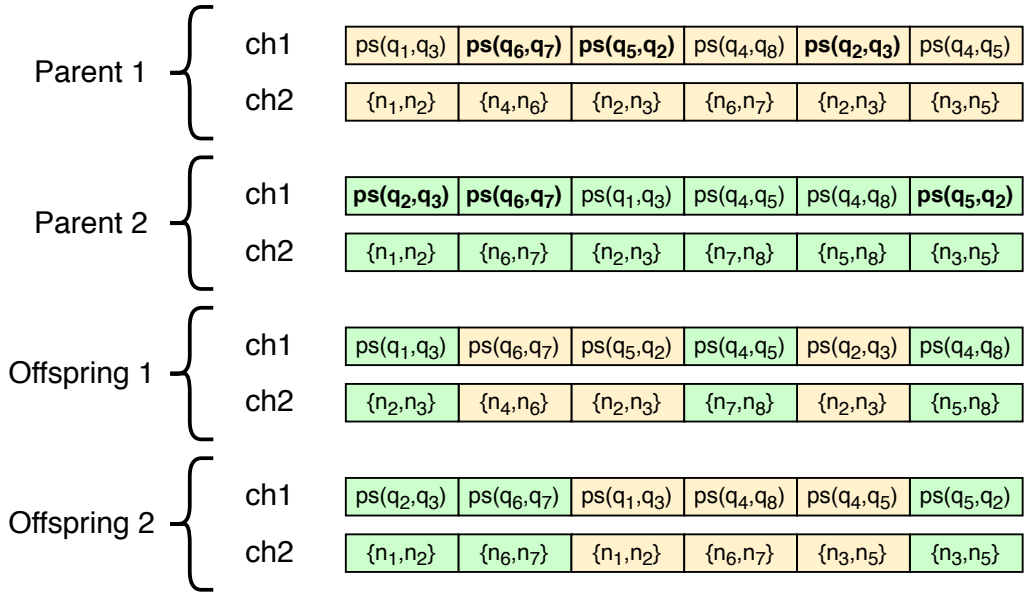


Figura 4.1: Dos cromosomas hijos son generados a partir de dos cromosomas padres mediante el operador de cruce.

posibles en el cromosoma. Por esta razón, optamos por utilizar una estrategia que minimiza el número de puertas *swap* en cada decisión de scheduling. De este modo, el esquema de codificación/decodificación propuesto no garantiza que al menos uno de los cromosomas factibles pueda codificar un schedule óptimo para las puertas p-s en la ronda p (después de un schedule para el subproblema definido por las rondas $1, \ldots, p-1$); en otras palabras, el espacio de búsqueda no es dominante.

Definimos $D(x, y)$ como la longitud del camino más corto entre los qubits x y y en el hardware cuántico QM, trivialmente, esta distancia es el mínimo número de puertas *swap* requeridas para mover el qstate en el qubit x al qubit y.

Sea $n(q)$ el qubit con el qstate q en un determinado instante de tiempo. Para mover los qstates q_i y q_j desde sus qubits actuales $n(q_i)$ and $n(q_j)$ hacia los qubits objetivo n_k y n_l elegimos la opción que minimiza el número global de puertas *swap*; esto es, q_i es movido hacia n_k y q_j es movido hacia n_l si $D(n(q_i), n_k) + D(n(q_j), n_l) < D(n(q_i), n_l) + D(n(q_j), n_k)$ y en otro caso se toman los movimientos alternativos. Los caminos elegidos son llamados *par de caminos mínimos*. El número de puertas *swap* requeridas viene dado por la longitud de estos caminos como está probado en los siguientes resultados.

Proposición 1. *Sean $n(q_i), \ldots, d(q_i)$ y $n(q_j), \ldots, d(q_j)$ un par caminos mínimos escogidos para mover los qstates en los qubits $n(q_i)$ y $n(q_j)$ hacia los qubits n_k y n_l, entonces el número de puertas swap requeridas es $D(n(q_i), d(q_i)) + D(n(q_j), d(q_j))$.*

Demostración. Si los dos caminos son disjuntos, es decir, no tienen ningún qubit en común, el resultado se demuestra de forma trivial en tanto que cada camino se traduce en un conjunto de puertas que son independientes de las puertas del otro camino. Si embargo, si los dos caminos tienen algún qubit en común, es posible que una puerta *swap* involucre dos qubits almacenando q_i y q_j, lo que puede impedir a uno de estos qubits de moverse con el mínimo número de swaps. Entonces, si este fuera el caso, en lugar de planificar una puerta swap que intercambie q_i y q_j, podemos

Input Una instancia P del problema QCCP con r rondas. El hardware cuántico QM. Un cromosoma $(sg_{p-1}, ch1, ch2)$. sg_{p-1} es un grafo solución del subproblema $1, \ldots, p$, $ch1$ es una permutación de puertas p-s en la ronda p. $ch2$ es una cadena de qubits conectados en QM de la misma longitud que $ch1$.
Output Un grafo solución sg_p para el subproblema $1, \ldots, p$ en el circuito QM

```
sg_p ← sg_{p-1}
for each p-s(q_i, q_j) en ch1 (y {n_k, n_l} en ch2) de izquierda a derecha do
    Sea path_i ← n(q_i) ⇝ d(q_i) y path_j ← n(q_j) ⇝ d(q_j) un par de caminos minimos desde
    {n(q_i), n(q_j)} hasta {n_k, n_l} en QM;
    while {n(q_i) ≠ d(q_i)} o {n(q_j) ≠ d(q_j)} do
        (n, n') ← (n(q_i), succ(n(q_i))) or (n(q_j), succ(n(q_j))) si es posible de modo que
        {n, n'} ≠ {n(q_i), n(q_j)};
        if {n, n'} ≠ {n(q_i), n(q_j)} then
            inserta una puerta swap en los qubits {n, n'} y actualiza sg_p;
            n ← n'; // avanza en path_1 o en path_2
        else
            intercambia en path_1 y path_2 los subcaminos desde los actuales qubits n y n'
            hacia sus qubits de destino, así que los nuevos caminos pasan a ser n, n' ⇝ d(n')
            y n' ⇝ d(n) si los antiguos eran n, n' ⇝ d(n) y n' ⇝ d(n') respectivamente;
        end if
    end while
    inserta una puerta p-s en los qubit {n_k, n_l} donde se almacenan los qstates {q_i, q_j} y
    actualiza sg_p;
end for
inserta una puerta mix en cada qubit almacenando un qstate;
return El grafo solución sg_p;
```

Algoritmo 1: Algoritmo de decodificación. Dado un cromosoma en la ronda p, este produce un grafo solución para el subproblema definido por las rondas $1, \ldots, p$. $succ(n(q_k))$ denota el sucesor del qubit $n(q_k)$ en $path_k$.

cambiar sus destinos y así la puerta *swap* no es necesaria y los qubits pueden ir a sus nuevos destinos sin swaps adicionales. □

El algoritmo 1 muestra el algoritmo de decodificación que se aplica al cromosoma $(sg_{p-1}, ch1, ch2)$ para obtener un schedule (un grafo solución sg_p) para el subproblema definido por las rondas $1, \ldots, p$. Este itera en las puertas p-$s(q_i, q_j)$ de $ch1$ y cada uno está planificado en un par de qubits $\{n_k, n_l\}$ en la misma posición de $ch2$. Para hacer esto, se empieza calculando un par de caminos mínimos $path_1$ y $path_2$ desde los qubits actuales de los qstates q_i y q_j hacia los qubits de destino n_k y n_l. Entonces, los arcos en estos caminos se consideran siguiendo el orden en cada camino, y por cada uno la puerta *swap* asociada es introducida en la solución parcial. En este proceso los arcos desde los dos caminos pueden ser intercalados tratando de evitar revertir dos qubits adyacentes almacenando q_i y q_j. Aunque, si en una iteración esta es la única opción entonces los caminos restantes desde los qstates hacia sus destinos son intercambiados en lugar de intercambiar los qubits. De este modo, el número de puertas *swap* puede mantenerse al mínimo. Después de las puertas p-s, las puertas *mix* son introducidas en cada qstate.

Input Una instancia de QCCP P con el conjunto de qstates Q, r rondas y los conjuntos de puertas $P\text{-}S_p$ y $SWAP_p$ en cada ronda $p \in 1, \ldots, r$. El hardware cuántico QM. El conjunto de $Parametros$ del MA: $popSize$, P_c, P_m
Output Un schedule H para la instancia P en el circuito QM.
Distribuir los qstates en Q en los qubits de QM;
$SG_0 \leftarrow$ conjunto de $popSize$ grafos de solución iniciales;
for p = 1 a r **do**
$SG_p \leftarrow MA(SG_{p-1}, P\text{-}S_p, MIX_p, Parameters)$;
end for
return El mejor schedule en SG_r;

Algoritmo 2: El Decomposition-Based Memetic Algorithm. Construye un schedule para una instancia de QCCP P iterando en las rondas $1, \ldots, r$ de P. En cada iteración construye schedules para la ronda p compatibles con las soluciones parciales del subproblema $1, \ldots, p-1$.

4.4. Estructura general de DBMA

La estructura general del DBMA que proponemos esta mostrada en el Algoritmo 2. Este tiene como entrada una instancia de QCCP con r rondas y un hardware cuántico QM, y produce un schedule H para P en QM. El algoritmo empieza distribuyendo los qstates en qubits; aquí asumimos que la distribución inicial es la misma para todas las soluciones candidatas. Entonces se calculan los grafos de solución iniciales SG_0. Estos grafos de solución son triviales de la forma que $g_{init} \rightarrow g_{end}$ que representan una solución de la ficticia ronda 0. El algoritmo itera un numero de rondas, y en cada iteración llama al algoritmo memético (MA) para extender las soluciones desde la ronda $p-1$ a la ronda p. DBMA devuelve el mejor grafo solución obtenido en la última ronda.

4.5. El Algoritmo Memético

El Algoritmo 3 muestra el MA propuesto para obtener soluciones en cada ronda p. Este empieza desde un conjunto de $popSize$ grafos de solución SG_{p-1} para los subproblemas $1, \ldots, p-1$ y produce un conjunto de grafos de solución SG_p del mismo tamaño para el subproblema $1, \ldots, p$. Inicialmente, este construye los $popSize$ cromosomas iniciales de la forma $(sg_{p-1}, ch1, ch2)$, para cada sg_{p-1} en SG_{p-1}; tomando en principio $ch1$ como permutaciones aleatorias de las puertas $p\text{-}s$ en la ronda p y $ch2$ como una secuencia de la misma longitud que $ch1$ de pares aleatorios de qubits conectados en QM (en Sección 4.5.1 podemos ver una alternativa heurística). Los cromosomas iniciales son evaluados para obtener el primer grafo solución candidato sg_p para cada uno. Cada cromosoma guarda su propio grafo solución sg_p a lo largo de las iteraciones del algoritmo.

Entonces, el MA itera hasta que se satisface la *Condición De Parada*. Esta condición es que no haya mejora en un numero dado de generaciones consecutivas. En cada iteración, los cromosomas en la población se organizan en pares aleatorios, que luego se cruzan y mutan. Los hijos resultantes se evalúan (Algoritmo 1) para obtener nuevos grafos de solución sg_p. Cada uno de estos grafos se somete a una búsqueda local y finalmente, en el paso de reemplazo, se seleccionan dos cromosomas mediante

Input Un conjunto SG_{p-1} de *popSize* grafos de solución para los subproblemas $1, \ldots, p-1$, los conjuntos de puertas $P\text{-}S_p$ y MIX_p. Los *parmetros*: *popSize*, P_c, P_m. El circuito cuántico QM.

Output Un conjunto SG_p de *popSize* grafos de solución para los subproblemas $1, \ldots, p$.

Población Inicial: Por cada $sg_{p-1} \in SG_{p-1}$ generado en el cromosoma inicial $(sg_{p-1}, ch1, ch2)$, donde $ch1$ es una permutación aleatoria de puertas en $P\text{-}S_p$ y $ch2$ es un conjunto de conexiones en QM, que puede ser calculado aleatoriamente o mediante algún heurístico;

Evaluación: El algoritmo de decodificación se aplica a los cromosomas iniciales para extender los grafos de solución sg_{p-1} a la primera serie de grafos de solución sg_p de acuerdo con los componentes $ch1$ y $ch2$ en cada cromosoma;

Búsqueda local: BL se aplica en los grafos de solución sg_p de la poblacion inicial;

while not *Condición de terminación* **do**

Selección: Los cromosomas en la población se barajan en pares aleatorios;

Recombinación: Cada par de cromosomas es cruzado y los hijos son mutados de acuerdo con las probabilidades P_c y P_m respectivamente;

Evaluación: El algoritmo de decodificación se aplica a los hijos para extender los grafos de solución sg_{p-1} a los nuevos grafos de solución sg_p;

Búsqueda local: BL se aplica en los grafos de solución sg_p de los hijos generados;

Reemplazo: Selección por torneo de dos cromosomas entre cada par de padres y sus dos hijos;

end while

$SG_p \leftarrow$ grafos de solución sg_p de los *popSize* cromosomas en la población;

return El conjunto de grafos de solución SG_p;

Algoritmo 3: Algoritmo memético. Construye una serie de schedules para la ronda p de una instancia P de QCCP después de las soluciones parciales para el subproblema definido por las rondas $1, \ldots, p-1$.

un torneo entre cada pareja de padres con sus respectivos hijos teniendo en cuenta sus grafos de solución sg_p.

Finalmente, el MA devuelve un conjunto de *popSize* grafos de solución SG_p que serán utilizados como valores de entrada en la siguiente iteración, si $p < r$, o que representan soluciones del problema completo, si $p = r$.

Además de lo anterior, los siguientes comentarios son necesarios para entender el MA.

Observación 1. *Una práctica común en los algoritmos meméticos es, después de una búsqueda local, recodificar en el cromosoma la solución mejorada. Esto se conoce como evolución Lamarckiana y permite a los cromosomas seguir aprendiendo características por lo que podrían transmitirlas a sus hijos en posteriores iteraciones. No podemos utilizar este tipo de evolución debido a la limitada expresividad del esquema de codificación, que está basado únicamente en permutaciones de puertas p-s y los qubits de destino de cada puerta. Por tanto, no es posible codificar el camino para mover cada puerta a sus qubits de destino. De esta forma, no es posible recodificar la estructura de un grafo solución en un cromosoma de modo que el algoritmo de decodificación pueda construir el mismo grafo solución. De hecho, esta es la principal razón para mantener los grafos de solución sg_{p-1} como un componente de los cromosomas de la ronda p.*

Observación 2. *El conjunto SG_p producido por el MA para la ronda p incluye el grafo solución sg_p de cada cromosoma de la anterior generación. Así, SG_p puede contener una serie de soluciones muy malas para el subproblema $1, \ldots, p$. Por lo tanto, se podría agregar una presión más selectiva al algoritmo eliminando varias de las peores soluciones en SG_p y, por ejemplo, reemplazarlas aleatoriamente por una mejor. Como es bien sabido, esto podría producir una convergencia prematura. Esta opción debería analizarse empíricamente.*

Observación 3. *La población inicial se genera aleatoriamente, lo que también puede dar lugar a muchas soluciones malas. Entonces, como sucede a menudo, podría ser bueno explotar algún método heurístico para diseñar todos o algunos de los cromosomas iniciales. Esto a menudo es bueno a riesgo de producir convergencia prematura también. Por lo tanto, también sería necesario un análisis empírico. En la sección 4.5.1, proponemos un método de inicialización heurística.*

4.5.1. Población inicial heurística

La generación aleatoria de los cromosomas iniciales puede proporcionar diversidad en el material genético de la población, pero a menudo sucede que los cromosomas aleatorios son tan malos que un algoritmo genético difícilmente puede converger en buenas soluciones incluso después de un gran número de generaciones. Por esta razón, podría ser mejor explotar algunas heurísticas para iniciar al menos una parte de la población con individuos que representen buenas soluciones. Esto puede ayudar a alcanzar mejores soluciones rápidamente, a riesgo de convergencia prematura. En cualquier caso, la población inicial heurística ha demostrado ser buena en algunos problemas conocidos como Travelling Salesman Problem. [13], donde se explota la heurística del vecino más cercano o el Job Shop Scheduling Problem [14], donde los autores utilizaron heurísticas de ordenamiento de variables y valores para algunos tipos específicos de instancias.

Proponemos utilizar un heurístico tipo vecino más cercano para construir el componente $ch2$ de los cromosomas. La idea es simple: las puertas p-s en $ch1$ son visitadas de izquierda a derecha y por cada puerta $p\text{-}s(q_i, q_j)$ se coloca uno de sus pares más cercanos de qubits adyacentes (n_k, n_l), considerando la noción de par mínimo de caminos como distancia entre pares de qubits; entonces los qubits de destino candidatos son de la forma

$$(n_k, n_l) = argmin\{min(D(n(q_i), n) + D(n(q_j), n'), \\ D(n(q_i), n') + D(n(q_j), n)), (n, n') \in E_{p-s}\} \quad (4.1)$$

Está claro que, en general, el destino candidato no es único, en ese caso uno de ellos se elige aleatoriamente. Además, podemos dar prioridad a los arcos que unen qubits no seleccionados previamente para favorecer la ejecución paralela de qgates. De esta manera, el procedimiento de inicialización puede obtener una diversidad de componentes heurísticos de $ch2$.

4.5.2. Búsqueda Local

Input Un grafo solución sg_p. Una estructura de vecindad $\mathcal{N}$
Output Un grafo solución (con suerte mejorado)
  while not *Condición de terminación* **do**
    $sg_p^* \leftarrow argmin\{C_{max}^*(sg_p'), sg_p' \in \mathcal{N}(H)\}$;
    if $C_{max}(sg_p^*) \leq C_{max}(sg_p)$ **then**
      $sg_p \leftarrow sg_p^*$;
    end if
  end while
  return sg_p;

Algoritmo 4: Algoritmo de búsqueda local. $C_{max}(sg)$ denota el makespan actual del grafo solución sg y $C_{max}^*(sg)$ es el makespan estimado del grafo solución vecino sg obtenido tras un movimiento. La *Condición de Terminación* se satisface cuando el mejor makespan estimado es peor o igual que el makespan de la solución actual o cuando el makespan real del vecino seleccionado es peor.

La búsqueda local esta implementada mediante una vecindad $\mathcal{N}$ de cada punto en el espacio de búsqueda como un conjunto schedules alcanzables por una regla de transformación dada. El algoritmo de búsqueda local itera en varios pasos hasta que se satisfaga alguna *Condición de parada*. En cada iteración, se selecciona una solución vecina si cumple un criterio dado; en nuestro caso, el vecino seleccionado es aquel con la estimación de makespan más baja y que además cuyo makespan estimado sea mejor que el makespan de la solución actual. Finalmente, el vecino seleccionado es aceptado de acuerdo con un segundo criterio: si el makespan real es mejor o igual que el makespan de la solución actual en nuestro caso. El grafo solución aceptado sustituye al antiguo en el cromosoma, y la búsqueda local continúa desde este cromosoma. La condición de terminación que consideramos aquí es que el vecino aceptado sea peor que el de la solución actual.

Observación 4. *Una alternativa es cambiar el criterio de selección de los vecinos, de modo que también se puedan seleccionar vecinos cuyo makespan estimado sea igual al makespan de la solución actual. Esto podría permitir que en iteraciones posteriores se encuentre un nuevo vecino que pueda mejorar el makespan, pero plantea varios problemas. Puede darse el caso de que se formen bucles infinitos, lo que podría solventarse con una condición de terminación que se cumpla tras una serie de iteraciones sin mejora. Esto requeriría un análisis empírico.*

El punto clave de un algoritmo de búsqueda local es la estructura de vecindad. Una de las contribuciones de este trabajo es diseñar estructuras eficientes para el QCCP. Esta cuestión se aborda en profundidad en la Sección 5.

Capítulo 5

Estructuras de vecindad

Como se mencionó, el punto clave de un algoritmo de búsqueda local eficiente son las estructuras de vecindad. Una buena vecindad proporciona un número limitado de soluciones candidatas obtenidas a partir de una solución dada realizando pequeños cambios en su estructura. Idealmente, estos cambios no deberían ser absolutamente ciegos, sino que deberían guiarse por algún tipo de conocimiento del problema para que los vecinos tengan una cierta probabilidad de ser mejores que la solución original. La evaluación de las soluciones vecinas puede ser costosa y, por lo tanto, a menudo se sustituye por algún método que proporcione una evaluación aproximada.

En esta sección proponemos algunas estructuras diseñadas específicamente para el QCCP. Estas estructuras están diseñadas a partir de la noción de camino crítico, como se hizo para muchos otros problemas de scheduling en las últimas décadas [15, 16, 10, 12]. Específicamente, las estructuras de vecindad que proponemos aquí se basan en movimientos en el camino crítico que producen schedules factibles y que tienen la oportunidad de producir alguna mejora. La estimación del makespan de los vecinos se realiza a partir de las nociones de cabezas y colas de las operaciones, que pueden evaluarse en tiempo constante después de un movimiento.

Las estructuras de vecindad propuestas, denominadas $\mathcal{N}_1$, $\mathcal{N}_2$ y $\mathcal{N}_3$ están descritas a continuación.

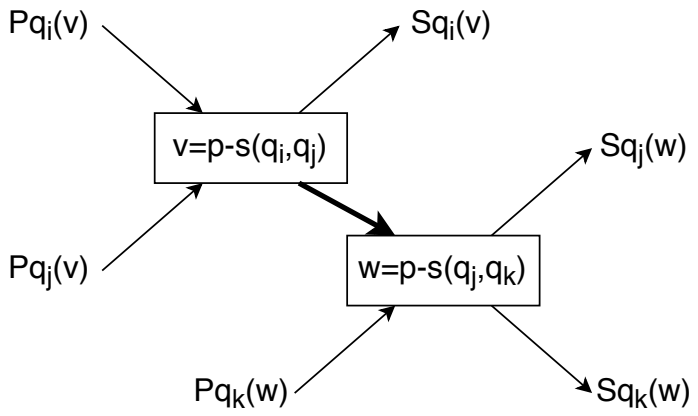
5.1. $\mathcal{N}_1$. Intercambia dos puertas p-s

La primera estructura consiste en invertir el arco entre dos puertas *p-s* consecutivas, v y w, en un camino crítico (deben compartir, por tanto, algún qstate). Esta inversión debe acompañarse de una reorganización de los predecesores y sucesores de las puertas involucradas en el grafo solución, de tal forma que se obtenga un schedule factible.

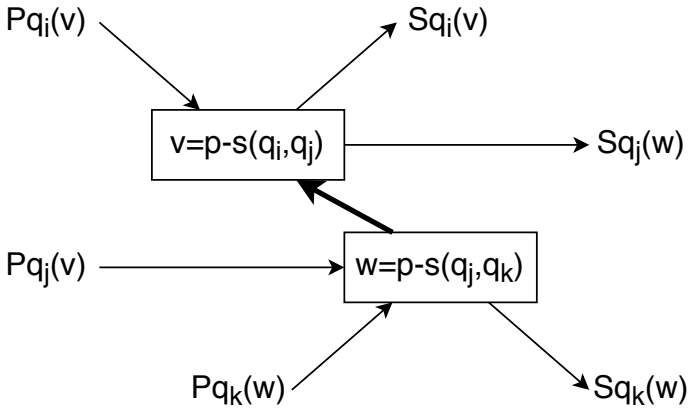
La figura 5.1 muestra un ejemplo de estos movimientos considerando las puertas $v = p\text{-}s(q_i, q_j)$ y $w = p\text{-}s(q_j, q_k)$ que comparten el qstate q_i. $P_{q_i}(v)$ representa a la puerta procesada por el qstate q_i justo antes de la puerta v, y $S_{q_i}(v)$ representa a la puerta procesada por el qstate q_i justo después de la puerta v.

El siguiente resultado garantiza la factibilidad del schedule vecino.

Proposición 2. *Si los nodos v y w son consecutivos en un camino crítico de un grafo solución, entonces no puede haber otro camino desde v hasta w en el grafo solución.*



(a) Antes del movimiento.



(b) Después del movimiento.

Figura 5.1: Ilustración de la estructura de vecindad $\mathcal{N}_1$. Un movimiento consiste en invertir un arco simple (v, w) en el camino crítico conectando dos puertas *p-s* y reorganizando los sucesores y predecesores de las dos puertas. El grafo solución resultante es acíclico y por tanto representa una solución factible.

Demostración. Considerando que las puertas $v = p\text{-}s(q_i, q_j)$ y $w = p\text{-}s(q_j, q_k)$ sean consecutivas en un camino crítico (ver Figura 5.1). Si existe otro camino desde v hasta w, este será de la forma v,$S_{q_i}(v)$,...,$P_{q_k}(w)$,w. Como el coste de los arcos es no-negativo y los arcos (v, w) y $(v, S_{q_i}(v))$ tienen el mismo coste, el coste del camino anterior sería mayor que el coste del arco (v, w), lo cual es una contradicción con el hecho de que (v, w) pertenece al camino crítico. □

Corolario 1. *Tras invertir el arco (v, w) no habrá ciclos en el grafo solución y por lo tanto representa un schedule factible.*

Por tanto, dado un grafo solución sg, la vecindad $\mathcal{N}_1(sg)$ es el conjunto completo de schedules que pueden obtenerse a partir de sg mediante la inversión de arcos en el camino crítico que conecten dos puertas *p-s*. Es fácil ver que estos arcos son los únicos que pueden dar lugar a una mejora del schedule después de una sola inversión. Claramente, invertir un solo arco que no pertenece a un camino crítico no puede dar lugar a ninguna mejora ya que el schedule resultante mantiene inalterado el camino crítico. Otra observación importante es que el camino crítico puede no ser único, por lo que podríamos considerar todos los caminos críticos en el cálculo de la vecindad. Sin embargo, este procedimiento podría ser computacionalmente costoso y, por lo tanto, normalmente solo se considera un camino crítico.

Para evaluar un schedule vecino, se debe calcular un camino crítico. Aunque se puede obtener un camino crítico en tiempo polinomial, hacerlo para todos los vecinos puede ser computacionalmente costoso. Por lo tanto, proponemos utilizar un método aproximado que se base en las nociones de *cabezas* y *colas*, lo que permite realizar estimaciones en tiempo constante.

Definimos la cabeza de una puerta v (denominada r_v) como camino con máximo coste desde el nodo g_{start} hasta v, mientras que la cola de v (denominada q_v) está definida como el camino con coste máximo desde v hasta g_{end} menos la duración de la propia puerta.

La estimación está basada en el procedimiento *lpath* para el clásico job shop scheduling problem propuesto en [17]. Después de invertir el arco arc $(v, w) = (p\text{-}s(q_i, q_j), p\text{-}s(q_j, q_k))$ en un schedule, las nuevas cabezas y colas para las operaciones v y w pueden estimarse de la forma siguiente:

$$
\begin{aligned}
r'_w &= \text{máx}\left\{r_{Pq_k(w)} + p_{Pq_k(w)}, r_{Pq_j(v)} + p_{Pq_j(v)}\right\} \\
r'_v &= \text{máx}\left\{r_{Pq_i(v)} + p_{Pq_i(v)}, r'_w + p_w\right\} \\
q'_v &= \text{máx}\left\{q_{Sq_i(v)} + p_{Sq_i(v)}, q_{Sq_j(w)} + p_{Sq_j(w)}\right\} \\
q'_w &= \text{máx}\left\{q_{Sq_k(w)} + p_{Sq_k(w)}, q'_v + p_v\right\}
\end{aligned}
$$

Por lo tanto, se puede estimar el makespan del nuevo grafo solución sg' como la longitud máxima de los caminos más largos desde el nodo *start* al nodo *end* a través de los nodos v y w respectivamente, es decir:

$$
C^*_{max}(S') = \text{máx}\left\{r'_w + p_w + q'_w, r'_v + p_v + q'_v\right\} \tag{5.1}
$$

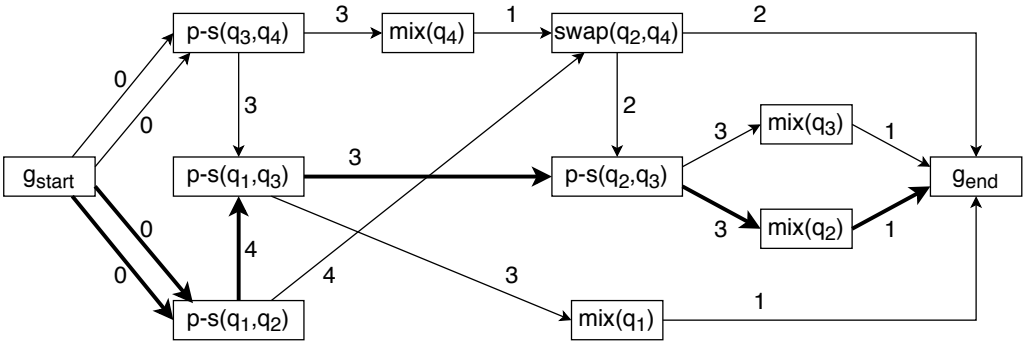
Nótese que esta estimación es, de hecho, un límite inferior en el makespan real de la solución vecina S', porque considera la longitud exacta de dos caminos del grafo resultante desde el nodo g_{start} y el nodo g_{end}, uno a través del nodo v y otro a través del nodo w.

Como ejemplo, la Figura 5.2(a) muestra un grafo solución obtenido por la inversión del arco crítico $(p\text{-}s(q_1, q_3), p\text{-}s(q_1, q_2))$ desde el grafo solución de la Figura 3.1(c). El diagrama de Gantt resultante de este schedule esta mostrado en la Figura 5.2(b). Podemos ver que la inversión fue capaz de reducir el makespan de 16 a 11 unidades de tiempo. El circuito cuántico compilado se muestra en la Figura 5.2(c).

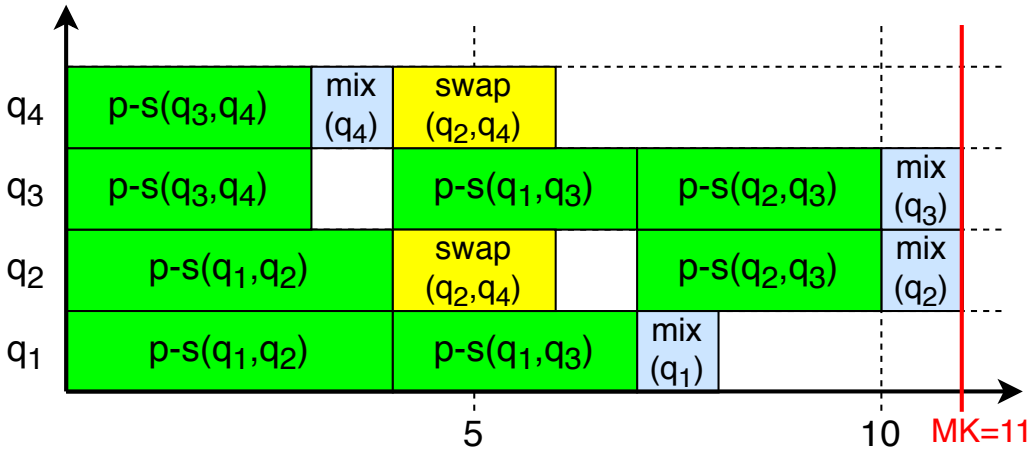
5.2. $\mathcal{N}_2$. Intercambia una puerta p-s y una puerta swap

Un movimiento en esta estructura consiste en invertir un arco (v, w), donde $v = p\text{-}s(q_i, q_j)$ y $w = swap(q_i, q_j)$ son dos puertas consecutivas en un camino crítico que comparten sus dos qstates (y por tanto estando conectadas por dos arcos). De forma similar a la anterior, los predecesores y sucesores de las puertas involucradas deben ser reorganizados; en este caso los predecesores y sucesores de v se convierten en predecesores y sucesores de w y viceversa. La Figura 5.3 muestra un ejemplo donde los arcos $(p\text{-}s(q_i, q_j), swap(q_i, q_j))$ son invertidos.

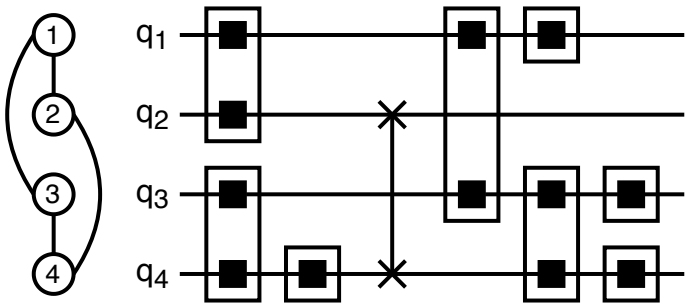
Utilizando un razonamiento similar al anterior, es fácil ver que los movimientos $\mathcal{N}_2$ producen schedules factibles. Sin embargo, no pueden mejorar ni empeorar el



(a) Grafo. El camino crítico está indicado con arcos en negrita y su longitud (o makespan) se reduce de 16 a 11.



(b) Diagrama de Gantt.



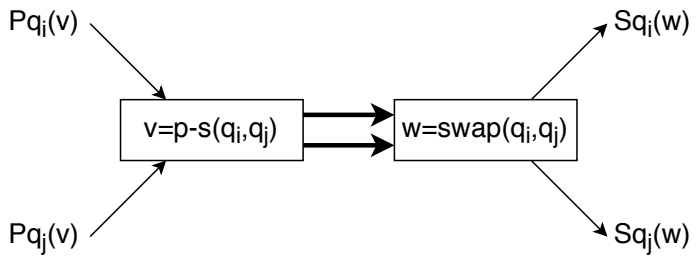
(c) El circuito cuántico compilado equivalente.

Figura 5.2: La solución obtenida tras invertir el arco crítico $(p{-}s(q_1, q_3), p{-}s(q_1, q_2))$ del grafo solución de la Figura 3.1(c).

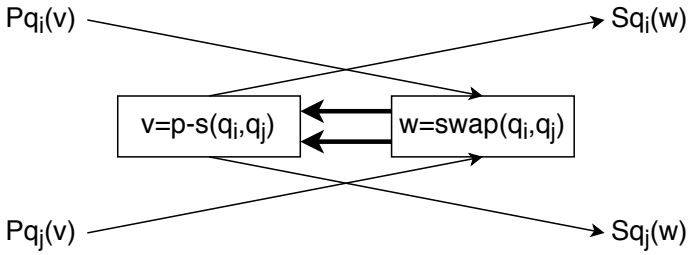
makespan actual, por lo que pueden no parecer útiles. Sin embargo, aún pueden ser interesantes si permiten realizar otro movimiento después. De hecho, esta es la razón para considerar la tercera estructura de vecindad.

5.3. $\mathcal{N}_3$. Un movimiento $\mathcal{N}_2$ seguido de un movimiento $\mathcal{N}_1$

Si en el camino crítico hay una secuencia $\cdots \rightarrow p\text{-}s(q_i, q_j) \rightarrow swap(q_i, q_j) \rightarrow p\text{-}s(q_i, q_k) \rightarrow \ldots$, consideramos aplicar un movimiento $\mathcal{N}_2$ seguido de uno $\mathcal{N}_1$, así



(a) Antes del movimiento.



(b) Después del movimiento.

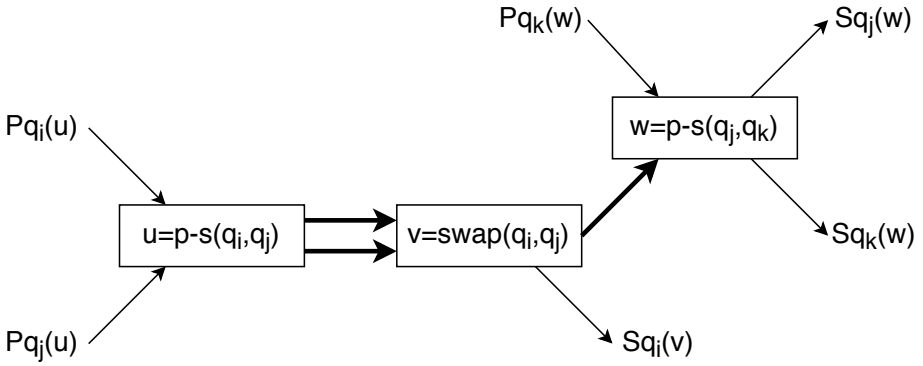
Figura 5.3: Ilustración de un movimiento $\mathcal{N}_2$. Este intercambia el orden de una puerta *p-s* y una puerta *swap* que comparten los mismos qstates.

que la secuencia final pasa a ser $\cdots \rightarrow swap(q_i, q_j) \rightarrow p\text{-}s(q_i, q_k) \rightarrow p\text{-}s(q_i, q_j) \rightarrow \ldots$. Se muestra un ejemplo en la Figura 5.4. Alternativamente si en el camino crítico hay una secuencia $(u = p - s(q_i, q_j), v = swap(q_j, q_k), w = p - s(q_j, q_k))$, esta se convierte en una secuencia $(w = p - s(q_j, q_k), u = p - s(q_i, q_j), v = swap(q_i, q_j))$ tras aplicar un movimiento $\mathcal{N}_2$ seguido de uno $\mathcal{N}_1$.

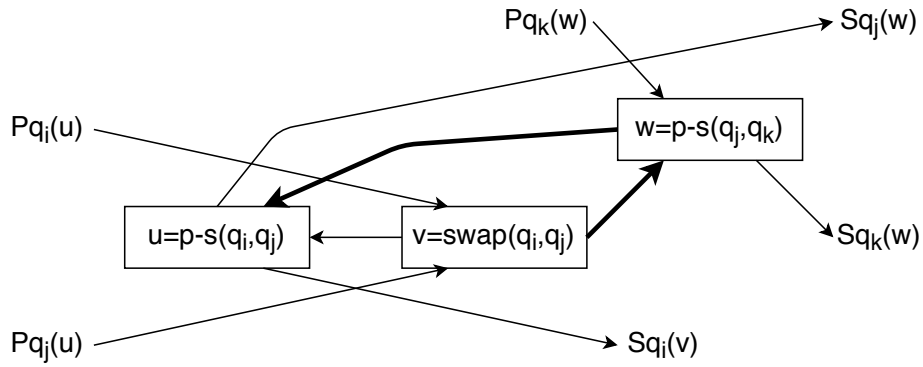
La factibilidad del nuevo schedule se deriva de las propiedades de $\mathcal{N}_1$ y $\mathcal{N}_2$. La estimación para esta vecindad en una extensión de la usada en la primera vecindad. Si una secuencia $(u = p - s(q_i, q_j), v = swap(q_i, q_j), w = p - s(q_j, q_k))$ se convierte en una secuencia $(v = swap(q_i, q_j), w = p - s(q_j, q_k), u = p - s(q_i, q_j))$, entonces las nuevas cabezas y colas para las puertas u, v y w se calculan como:

$$
\begin{aligned}
r'_v &= \text{máx}\left\{r_{Pq_i(u)} + p_{Pq_i(u)}, r_{Pq_j(u)} + p_{Pq_j(u)}\right\} \\
r'_w &= \text{máx}\left\{r_{Pq_k(w)} + p_{Pq_k(w)}, r'_v + p_v\right\} \\
r'_u &= r'_w + p_w \\
q'_u &= \text{máx}\left\{q_{Sq_i(v)} + p_{Sq_i(v)}, q_{Sq_j(w)} + p_{Sq_j(w)}\right\} \\
q'_w &= \text{máx}\left\{q_{Sq_k(w)} + p_{Sq_k(w)}, q'_u + p_u\right\} \\
q'_v &= q'_w + p_w
\end{aligned}
$$

También puede ocurrir que en el grafo solución resultante haya una puerta swap justo después de una puerta mix. Si $Pq_i(u)$ es una puerta mix y su inicio es posterior a que termine la ejecución de $Pq_j(u)$ $(r_{Pq_j(u)} + p_{Pq_j(u)} < r_{Pq_i(u)})$ podemos intercambiar sus posiciones de acuerdo con lo explicado en la Sección 4.3 en tanto que puede mejorar el makespan, esto debe tenerse en cuenta a la hora de hacer la estimación.



(a) Antes del movimiento.



(b) Después del movimiento

Figura 5.4: Ilustración de un movimiento $\mathcal{N}_3$. Consiste en un movimiento $\mathcal{N}_2$ seguido de un movimiento $\mathcal{N}_1$.

$$
\begin{aligned}
r'_v &= \max\{r_{Pq_i(u)} + p_{Pq_i(u)}, r_{Pq_j(u)} + p_{Pq_j(u)}\} \\
r'_w &= \max\{r_{Pq_k(w)} + p_{Pq_k(w)}, r'_v + p_v\} \\
r'_u &= r'_w + p_w \\
q'_u &= \max\{q_{Sq_i(v)} + p_{Sq_i(v)}, q_{Sq_j(w)} + p_{Sq_j(w)}\} \\
q'_w &= \max\{q_{Sq_k(w)} + p_{Sq_k(w)}, q'_u + p_u\} \\
q'_v &= q'_w + p_w - 1
\end{aligned}
$$

Alternativamente si $(u = p - s(q_i, q_j), v = swap(q_j, q_k), w = p - s(q_j, q_k))$ se convierte en una secuencia $(w = p - s(q_j, q_k), u = p - s(q_i, q_j), v = swap(q_i, q_j))$:

$$
\begin{aligned}
r'_w &= \max\{r_{Pq_k(w)} + p_{Pq_k(w)}, r_{Pq_j(u)} + p_{Pq_j(u)}\} \\
r'_u &= \max\{r_{Pq_i(u)} + p_{Pq_i(u)}, r'_w + p_w\} \\
r'_v &= r'_u + p_u \\
q'_v &= \max\{q_{Sq_j(w)} + p_{Sq_j(w)}, q_{Sq_k(w)} + p_{Sq_k(w)}\} \\
q'_u &= \max\{q_{Sq_i(u)} + p_{Sq_i(u)}, q'_v + p_v\} \\
q'_w &= q'_u + p_u
\end{aligned}
$$

Por tanto, el makespan del nuevo schedule S puede estimarse como la máxima longitud de los caminos más largos desde el nodo *start* hasta el nodo *end* a través

de los nodos u, v y w respectivamente:

$$EC^*(S') = \max\{r'_w + p_w + q'_w, r'_v + p_v + q'_v, r'_u + p_u + q'_u\}, \tag{5.2}$$

que es una cota inferior en el makespan de la solución vecina S'.

Nótese que en el caso de que la secuencia sea $(u = p - s(q_i, q_j), v = swap(q_j, q_k), w = p - s(q_j, q_k))$ su transformación no puede mejorar el makespan actual, solo puede empatarlo o empeorarlo, por lo que puede no parecer útil. Sin embargo, aún puede ser interesante si consideramos la Anotación 4 de la Subsección 4.5.2 ya que podría permitir realizar otro movimiento después.

Capítulo 6

Presupuesto

En esta sección se especifican los costes asociados a la elaboración del proyecto. El presupuesto del proyecto viene dado tanto por el coste de material empleado y licencias de software como del salario del personal encargado. El valor de las licencias del software utilizado presentes en este presupuesto coincide con el precio de venta al público suministrado por cada uno de los proveedores[18] [19]. Por tanto, todos los valores de coste especificados se encuentran debidamente referenciados. La tabla muestra el presupuesto del proyecto para los recursos utilizados para llevar a cabo el desarrollo del proyecto.

Tabla 6.1: Costes directos

Elemento	Tipo de recurso	Tipo de Unidad	Unidades	Precio por unidad	Costo
Equipo	Material	Unidad	1	600 €	600 €
Personal investigador	Personal	Jornada laboral	12	1.867,41 €	22.408,92 €
Windows 10 Pro	Software	Licencia	1	259 €	259 €
Visual Studio Professional 2019	Software	Licencia	1	641 €	641 €
TOTAL					24.008,92 €

Los costes directos del proyecto son en total 24.008,92 €. A esto habría que sumarle otros costes indirectos como el consumo eléctrico y los materiales de oficina.

Sobre el presupuesto de ejecución debe aplicarse los porcentajes de Gastos Generales (13 %) y Beneficio Industrial (6 %). En la tabla se muestra el siguiente Presupuesto Base de Licitación sin IVA que asciende a la cantidad de 28.570,61 €.

Sobre el presupuesto anterior debe aplicarse IVA (21 %). En la tabla se muestra el siguiente Presupuesto Base de Licitación con IVA.

Tabla 6.2: Presupuesto base

TOTAL EJECUCIÓN DEL PROYECTO	**24.008,92 €**
Beneficio industrial 6 %	1.440,54 €
Gastos generales 13 %	3.121,16 €
PRESUPUESTO BASE SIN IVA	**28.570,61 €**
IVA 21 %	5.999,83 €
PRESUPUESTO BASE CON IVA	**34.570,44 €**

El presupuesto Base de Licitación con IVA del proyecto asciende a la cantidad de 34.570,44 €.

Capítulo 7

Planificación

En la Figura 7.1 se muestra un diagrama Gantt con la planificación del proyecto. Esta planificación está dividida en varias partes, correspondientes a fase de desarrollo del proyecto.

- **Análisis:** Comienza con el estudio del problema y el análisis del estado del arte. En esta fase, se plantean una serie de alternativas y se hace una planificación de tareas a realizar.
- **Planificador y esquema de codificación:** La primera de estas tareas es diseñar un planificador o generador de schedules y una forma de codificarlos. Posteriormente viene su implementación.
- **Algoritmo genético:** Tras esto diseñamos el algoritmo genético, utilizando el esquema de codificación previo como cromosoma y el planificador para obtener los makespan de estos. Aquí se incluyen los operadores de mutación, cruce y diversificación que actúan sobre los cromosomas.
- **Búsqueda local:** Una vez que tenemos el algoritmo genético procedemos a diseñar el algoritmo de búsqueda local. Para ello primero diseñamos una serie de estructuras de vecindad, además de la forma de estimar los makespan y como se seleccionan los vecinos. Se implementa esta búsqueda local y a continuación se realiza el primer estudio experimental. En este estudio experimental se selecciona una configuración de parámetros lo mejor posible y se lanza una serie de ejecuciones para cada instancia calculando entre otros la media de sus makespan.
- **DBMA:** Se diseña este DBMA y se implementa. Posteriormente tras otro estudio experimental como el mencionado anteriormente.
- **Heuristico inicial:** Se diseña el heurístico para la generación de los individuos iniciales de una población.
- **Desarrollo documentación:** Paralelamente a todo el proyecto se redacta la documentación del proyecto, que consta de un paper y de la memoria de este TFM.
- **Extensiones del problema:** Finalmente están planificadas las extensiones del problema, QCCP-V y QCCP-X. En esta fase se diseñarán los algoritmos

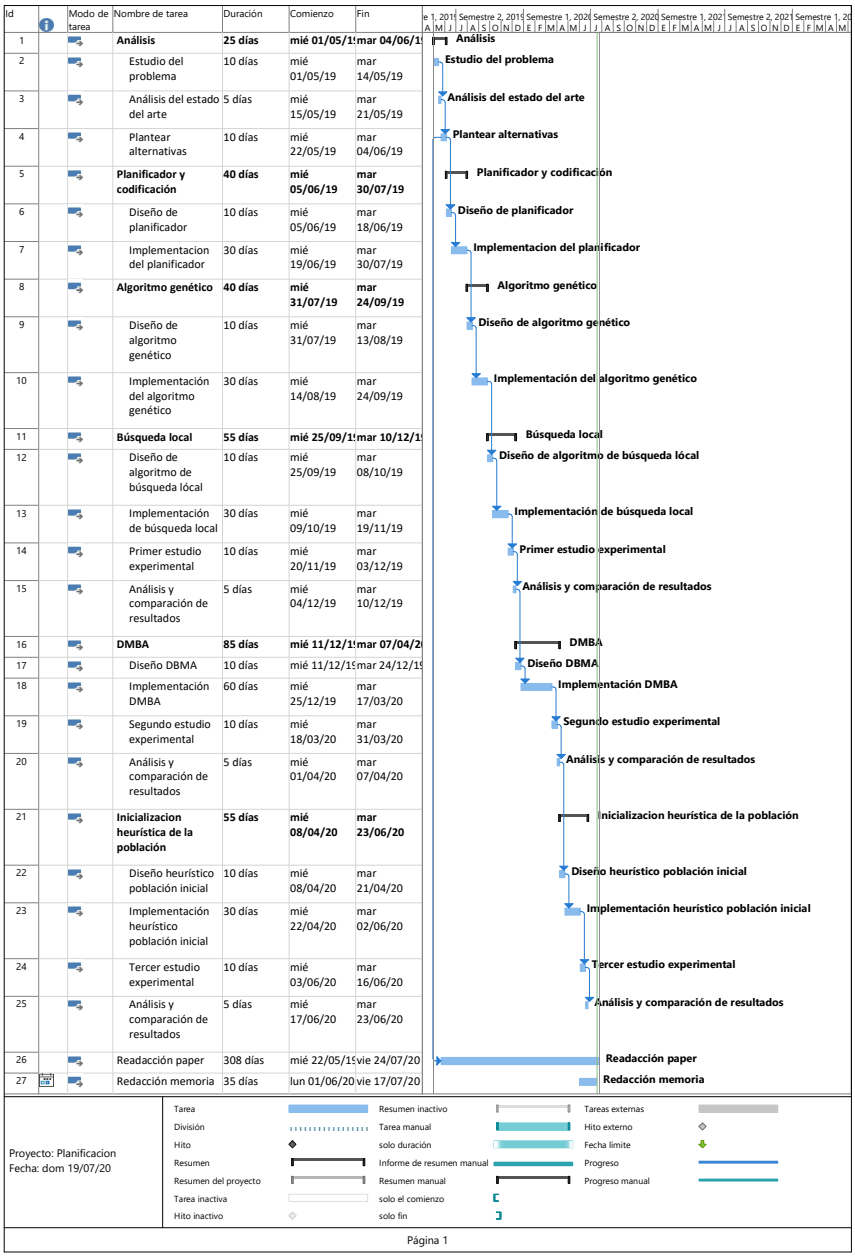


Figura 7.1: Diagrama de Gantt con la planificación del proyecto.

correspondientes y se realizara su implementación. Está previsto desarrollar nuevas estructuras de vecindad específicas de estas variantes del problema y la redacción de un nuevo paper con sus resultados.

Capítulo 8

Estudio experimental

Nuestros métodos están implementados en C++ y se ejecutan en un ordenador con 16 GB de RAM y un procesador Intel Core i5-7400 CPU de 3.00 GHz. Como hemos propuesto un método estocástico, lanzaremos 10 ejecuciones por instancia con el fin de obtener resultados estadísticamente significantes. Para cada instancia proporcionamos cual es el mejor makespan obtenido en las 10 ejecuciones, la mejor solución promedio, la peor de las mejores soluciones y el tiempo promedio de ejecución.

8.1. Benchmarks considerados

En este estudio experimental utilizamos como benchmark una serie de instancias tomadas de [2], que pueden ser descargadas desde la web[1]. El benchmark incluye instancias de distintos tamaños dependiendo del número de qubits del chip cuántico considerado ($N = \{4, 8, 21, 40\}$ (ver Figura 2.1)). Para cada tamaño de chip, se consideran diferentes niveles de utilización ($u = \{0.9, 1.0\}$) y un número requerido de pasadas de compilación ($P = \{1, 2\}$). El benchmark consta de 50 instancias para cada combinación de $\{N, P, u\}$, donde cada instancia es representativa de un grafo G a ser particionado por el procedimiento *MaxCut* a realizar. En este estudio experimental solo se están teniendo en cuenta las instancias pertenecientes a los chips cuánticos de tamaño $N = \{8, 21, 40\}$, con un nivel de utilización de $u = 1.0$ y un numero de pasadas de pasadas de compilación $P = 2$ ya que se consideran las más difíciles. Por ejemplo, en la Figura 8.1 se muestra una solución obtenida en la instancia 2 para el chip de tamaño $N = 8$ con utilización de $u = 1.0$ y un numero de pasadas de pasadas de compilación $P = 2$ que tiene un makespan menor que la mejor solución obtenida por Chand [5].

8.2. Análisis de la búsqueda local propuesta

Comenzamos haciendo un estudio experimental preliminar para probar la eficacia de la búsqueda local propuesta. Para empezar, hemos comprobado que aproximadamente el 80 % de vecinos elegidos (es decir, que su makespan estimado debe ser menor que el makespan del individuo actual), son de la vecindad $\mathcal{N}_1$ y el 20 % son

[1] `https://ti.arc.nasa.gov/m/groups/asr/planning-and-scheduling/VentCirComp17_data.zip`

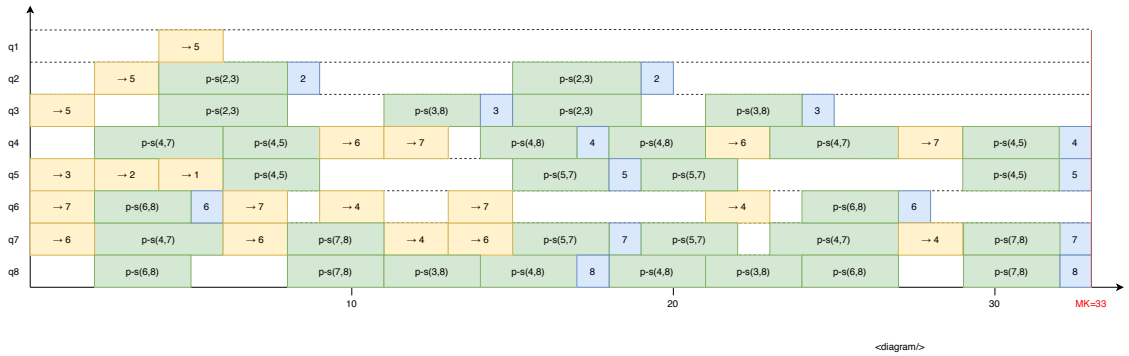


Figura 8.1: Representación Gantt de una solución relativa a la instancia 2 perteneciente al conjunto N8u1.0P2 del benchmark.

de la vecindad $\mathcal{N}_3$. En la tabla 8.1 también se puede ver que, de los vecinos elegidos, la mayoría son en verdad mejores.

Tabla 8.1: Porcentajes de cada vecino

Instancias N8				
Vecindad	total	mejora	empata	empeora
$\mathcal{N}_1$	85	81	4	0
$\mathcal{N}_3$	15	12	2	1
Instancias N21				
Vecindad	total	mejora	empata	empeora
$\mathcal{N}_1$	83	75	9	0
$\mathcal{N}_3$	17	13	3	1
Instancias N40				
Vecindad	total	mejora	empata	empeora
$\mathcal{N}_1$	77	67	10	0
$\mathcal{N}_3$	23	18	4	1

En la tabla se muestra el porcentaje de cada tipo de vecindad entre vecinos elegidos.

En la Tabla 8.2 se muestran los resultados comparados de utilizar búsqueda local o no. Podemos comprobar que utilizando la búsqueda local el algoritmo se comporta mejor, en general, tanto en media como en el mejor makespan obtenido.

8.3. Parametros de ejecución

Los parámetros de entrada para los experimentos realizados son los siguientes. El tamaño de la población es de 1000 para las versiones de N21 y N40 y de 200 para las de N8. Se utiliza una mutación con probabilidad de 5 por ciento y 3 de máximo de genes mutados. Se estable un criterio de parada a las 200 generaciones sin mejora para cada fase del algoritmo. Este criterio de parada se ha seleccionado para que el tiempo de ejecución del algoritmo sea similar al utilizado por Chand [5], de modo que se puedan comparar en igualdad de condiciones. Cada 10 generaciones sin mejora se produce una diversificación con un máximo de 5 mutaciones por cromosoma.

La Figura 8.2 y la Figura 8.3 muestran la curva de convergencia para una ejecución de la instancia 1 de N40 utilizando o no el operador de diversificación respectivamente. Podemos observar que en el primer caso el makespan medio de la población

Tabla 8.2: Resultados comparados con y sin búsqueda local.

	Sin búsqueda local				Con búsqueda local			
Instancia	Mejor	Peor	Media	Tiempo	Mejor	Peor	Media	Tiempo
Instancias N8								
1	**37**	37	**37.00**	2.1	**37**	37	**37.00**	2.2
2	**33**	37	**34.70**	2.6	**33**	37	34.90	2.6
3	**32**	34	**32.20**	2.4	**32**	33	**32.20**	2.2
4	**32**	34	**33.40**	2.4	33	34	**33.40**	2.0
5	**28**	30	29.00	2.4	**28**	28	**28.00**	2.3
6	34	35	**34.40**	2.3	**33**	35	**34.40**	2.4
7	**30**	31	30.90	2.3	**30**	30	**30.00**	2.2
8	**33**	34	**33.80**	2.5	**33**	34	**33.80**	2.3
9	**35**	36	**35.60**	2.5	**35**	37	35.90	2.5
10	**34**	38	**35.40**	2.6	**34**	38	36.10	2.8
#Mejores	9		8		9		7	
Instancias N21								
1	**46**	50	**47.70**	52.7	**46**	51	48.10	48.1
2	**47**	49	**48.30**	47.0	**47**	52	48.60	42.4
3	**45**	50	47.10	52.1	**45**	49	**46.90**	40.7
4	**43**	49	46.10	50.6	**43**	48	**45.90**	48.5
5	**48**	53	**50.50**	54.4	50	55	52.30	49.1
6	49	55	51.90	46.8	**46**	53	**51.50**	45.3
7	**51**	63	55.10	50.2	52	59	**53.90**	51.2
8	**45**	51	47.80	47.8	**45**	47	**46.30**	46.6
9	51	55	52.50	46.9	**50**	54	**52.00**	48.2
10	**52**	60	55.00	55.2	**52**	60	**54.30**	50.4
#Mejores	8		3		8		7	
Instancias N40								
1	**59**	72	64.00	157.6	**59**	67	**63.00**	149.1
2	69	83	74.10	216.6	**66**	81	**71.70**	167.4
3	70	76	72.90	177.5	**65**	80	**70.30**	164.1
4	73	87	78.40	206.0	**65**	84	**75.10**	180.5
5	64	76	72.00	202.7	**63**	78	**70.90**	191.4
6	**71**	88	78.70	219.5	**71**	85	**77.50**	179.9
7	**61**	74	**68.90**	190.2	68	79	72.30	189.5
8	62	74	69.60	181.4	**61**	73	**67.50**	161.5
9	**63**	72	**68.40**	192.4	64	75	69.50	165.4
10	**69**	89	77.70	183.4	71	84	**76.30**	173.5
#Mejores	5		2		7		8	

Los valores en **negrita** son la mejor media de cada instancia y la mejor solución encontrada.

mantiene cierta distancia con la mejor solución encontrada, por lo que la diversidad de la población es mayor; mientras que en el segundo caso la media se acerca mucho al mejor makespan encontrado, por lo que podemos inferir que la diversidad de la población es escasa y esto puede conducir a una convergencia prematura.

8.4. Comparación con el estado del arte

En las Tablas 8.3, 8.4 y 8.5 se muestran los resultados obtenidos por el algoritmo con los parámetros descritos en la Sección 8.3 comparados con los resultados obtenidos por Chand [5], que son los mejores publicados en el estado del arte. Nuestro algoritmo se ejecuta 10 veces por instancia. Podemos observar que el algoritmo desarrollado en este trabajo obtiene unos resultados generalmente mejores que de

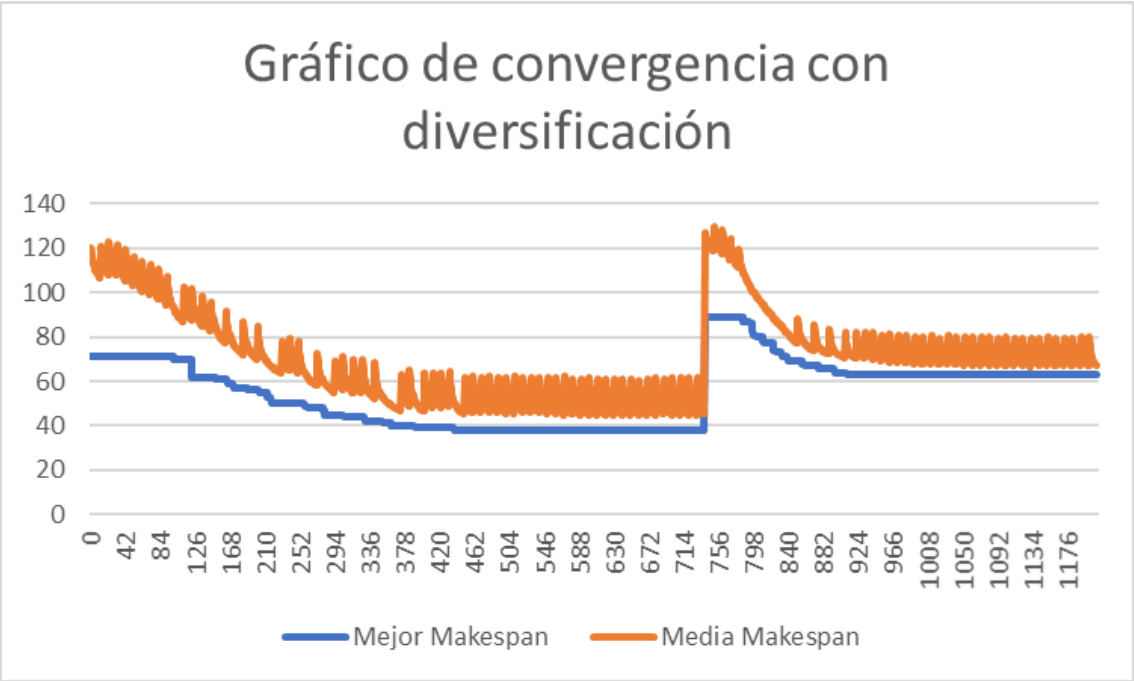


Figura 8.2: Curva de convergencia de una ejecución de la instancia 1 de N40 utilizando el operador de diversificación.

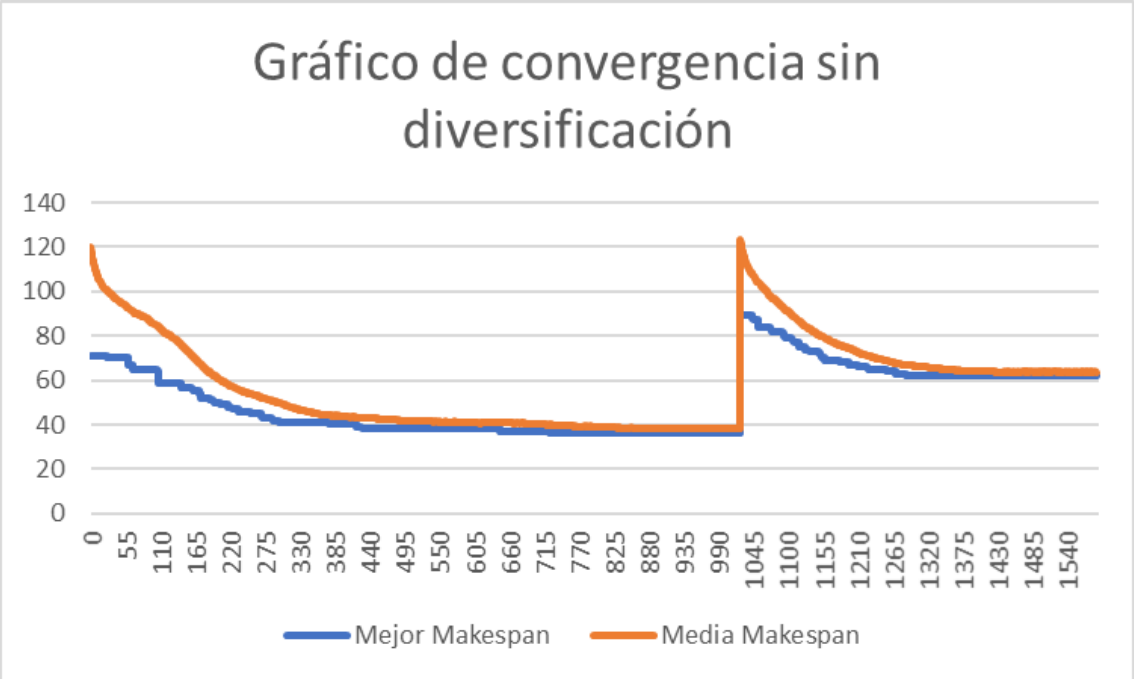


Figura 8.3: Curva de convergencia de una ejecución de la instancia 1 de N40 sin utilizar el operador de diversificación.

Chand [5], especialmente en las instancias de N=40. En el caso de las instancias de N=8 la media de las mejores soluciones de Chand [5] es menor en un mayor número de instancias, sin embargo, nuestro algoritmo obtiene más veces soluciones con el menor makespan encontrado. Para las instancias de N=21 nuestro algoritmo tiende a encontrar mejores soluciones tanto en media como en la mejor de todas aunque

sin demasiada diferencia. En el caso de las instancias de N=40 nuestro algoritmo tiene la mejor media en la mayoría de las instancias (43 de 50) y el mejor makespan encontrado solo es superado en única instancia por Chand [5].

Tabla 8.3: Resultados detallados en instancias con N=8.

	Chand [5]			Nuestro			
Instancia	Mejor	Peor	Media	Mejor	Peor	Media	Tiempo
1	**35**	35	**35.00**	37	37	37.00	2.2
2	36	36	36.00	**33**	37	**34.90**	2.6
3	**31**	31	**31.00**	32	33	32.20	2.2
4	**32**	34	33.80	33	34	**33.40**	2.0
5	**27**	27	**27.00**	28	28	28.00	2.3
6	35	35	35.00	**33**	35	**34.40**	2.4
7	31	31	31.00	**30**	30	**30.00**	2.2
8	34	34	34.00	**33**	34	**33.80**	2.3
9	**35**	35	**35.00**	**35**	37	35.90	2.5
10	38	38	38.00	**34**	38	**36.10**	2.8
11	**38**	38	**38.00**	39	41	39.70	2.7
12	33	33	**33.00**	**32**	37	34.70	2.5
13	**32**	32	**32.00**	**32**	33	32.10	2.4
14	**32**	32	**32.00**	**32**	32	**32.00**	2.1
15	35	36	35.53	**34**	35	**34.10**	2.5
16	**32**	32	**32.00**	**32**	32	**32.00**	2.1
17	36	36	36.00	**32**	32	**32.00**	2.5
18	**29**	30	**29.20**	30	31	30.20	2.2
19	**32**	32	**32.00**	**32**	32	**32.00**	2.3
20	31	32	31.13	**30**	30	**30.00**	2.6
21	27	27	27.00	**26**	28	**26.90**	2.1
22	39	39	**39.00**	**38**	41	40.30	2.6
23	**35**	37	**36.00**	36	37	36.80	2.6
24	**32**	32	**32.00**	33	35	33.20	2.3
25	**38**	39	38.27	**38**	38	**38.00**	2.6
26	29	29	29.00	**28**	29	**28.10**	2.2
27	**34**	34	**34.00**	**34**	37	35.90	2.5
28	32	32	32.00	**31**	31	**31.00**	2.3
29	**35**	37	**35.13**	37	38	37.10	2.9
30	31	32	**31.20**	**30**	32	31.30	2.2
31	**32**	32	**32.00**	**32**	34	33.10	2.1
32	**35**	37	**35.93**	**35**	38	36.30	2.5
33	42	42	**42.00**	**41**	44	42.70	2.7
34	35	35	35.00	**33**	34	**33.90**	2.5
35	**38**	38	**38.00**	39	39	39.00	2.3
36	**28**	29	**28.20**	30	32	31.30	2.3
37	**35**	35	**35.00**	**35**	37	35.40	2.6
38	29	30	29.93	**28**	30	**29.20**	2.6
39	30	30	30.00	**28**	32	**29.80**	2.4
40	**37**	38	**37.13**	**37**	39	38.40	2.5
44	**35**	35	**35.00**	36	38	36.80	2.6
42	33	34	33.07	**31**	33	**31.50**	2.3
43	**32**	34	33.07	**32**	32	**32.00**	2.2
44	39	41	**39.93**	**38**	43	41.20	3.0
45	38	38	38.00	**36**	37	**36.40**	2.3
46	**34**	34	**34.00**	**34**	36	34.20	2.3
47	38	39	38.80	**36**	38	**37.50**	2.4
48	**33**	33	**33.00**	**33**	33	**33.00**	2.1
49	**36**	37	**36.13**	**36**	38	37.80	2.9
50	**30**	32	31.27	31	31	**31.00**	2.2
#Mejores	28		29	37		25	

Los valores en **negrita** son la mejor media de cada instancia y la mejor solución encontrada.

Tabla 8.4: Resultados detallados en instancias con N=21.

	Chand [5]			Nuestro			
Instancia	Mejor	Peor	Media	Mejor	Peor	Media	Tiempo
1	49	53	51.20	**46**	51	**48.10**	48.1
2	50	55	52.87	**47**	52	**48.60**	42.4
3	**42**	48	**45.70**	45	49	46.90	40.7
4	44	48	**45.60**	**43**	48	45.90	48.5
5	52	54	53.00	**50**	55	**52.30**	49.1
6	50	55	52.87	**46**	53	**51.50**	45.3
7	55	57	56.47	**52**	59	**53.90**	51.2
8	49	52	50.40	**45**	47	**46.30**	46.6
9	54	56	54.40	**50**	54	**52.00**	48.2
10	54	60	56.70	**52**	60	**54.30**	50.4
11	47	51	48.87	**43**	48	**46.20**	41.3
12	56	62	58.67	**52**	57	**54.50**	47.3
13	**43**	45	44.40	**43**	44	**43.50**	38.1
14	**46**	48	**46.20**	47	58	51.60	51.0
15	46	48	47.40	**42**	46	**43.30**	40.1
16	57	64	60.33	**53**	58	**55.60**	50.1
17	**50**	52	**50.73**	51	56	53.40	47.4
18	54	61	59.40	**51**	60	**53.80**	47.1
19	56	62	59.27	**49**	56	**52.50**	45.9
20	50	53	51.80	**49**	54	**51.00**	47.5
21	**51**	54	**51.73**	52	57	54.50	49.3
22	54	55	54.53	**52**	55	**53.10**	46.9
23	**48**	51	**50.13**	**48**	53	50.60	47.9
24	50	55	53.73	**47**	57	**51.80**	49.4
25	**50**	53	**50.67**	53	59	56.10	45.2
26	46	52	47.80	**43**	48	**45.40**	43.2
27	61	66	62.67	**54**	66	**59.60**	53.5
28	47	51	49.80	**44**	51	**48.20**	48.0
29	**47**	50	**48.60**	48	54	50.20	41.5
30	53	56	55.00	**51**	57	**53.30**	50.1
31	**52**	57	**55.47**	55	61	58.30	48.7
32	52	52	52.00	**46**	51	**47.80**	45.8
33	52	57	55.80	**48**	53	**49.80**	45.1
34	**51**	56	**53.73**	54	60	56.60	45.4
35	**45**	51	**48.60**	47	52	49.70	44.5
36	**49**	55	**52.27**	52	60	55.00	50.1
37	**51**	54	52.93	**51**	54	**52.80**	44.4
38	53	55	**53.60**	**51**	58	54.40	51.1
39	**50**	55	**52.13**	53	62	59.00	43.3
40	**48**	57	52.87	**48**	54	**51.10**	43.6
41	**49**	53	50.93	**49**	49	**49.00**	35.6
42	50	53	**51.00**	**49**	53	51.10	44.3
43	**47**	51	**48.53**	48	52	50.20	43.7
44	**47**	52	**49.40**	48	52	**49.40**	39.9
45	**40**	47	**43.93**	42	48	45.70	49.2
46	**42**	46	**44.27**	**42**	51	44.90	51.0
47	52	53	**52.13**	**49**	54	52.50	47.9
48	**43**	47	**46.40**	46	50	47.20	42.4
49	54	58	**57.27**	**53**	61	57.30	48.6
50	53	57	54.67	**49**	53	**51.10**	47.2
#Mejores	21		22	35		29	

Los valores en **negrita** son la mejor media de cada instancia y la mejor solución encontrada.

Tabla 8.5: Resultados detallados en instancias con N=40.

	Chand [5]			Nuestro			
Instancia	Mejor	Peor	Media	Mejor	Peor	Media	Tiempo
1	65	70	69.30	**59**	67	**63.00**	149.1
2	74	77	75.90	**66**	81	**71.70**	167.4
3	71	77	74.50	**65**	80	**70.30**	164.1
4	74	82	77.30	**65**	84	**75.10**	180.5
5	78	78	78.00	**63**	78	**70.90**	191.4
6	81	83	82.10	**71**	85	**77.50**	179.9
7	79	83	80.80	**68**	79	**72.30**	189.5
8	68	77	73.70	**61**	73	**67.50**	161.5
9	66	67	**66.90**	**64**	75	69.50	165.4
10	80	80	80.00	**71**	84	**76.30**	173.5
11	68	72	70.54	**63**	72	**68.20**	166.4
12	74	77	75.60	**65**	79	**72.70**	165.0
13	62	65	64.53	**59**	72	**64.00**	151.0
14	74	84	78.87	**69**	80	**73.90**	173.8
15	78	78	78.00	**60**	79	**70.70**	164.4
16	77	78	77.93	**66**	77	**70.00**	184.7
17	78	78	78.00	**67**	83	**73.40**	185.3
18	79	82	80.40	**70**	79	**75.60**	170.7
19	70	71	70.93	**59**	71	**64.40**	150.7
20	78	85	81.53	**75**	90	**80.20**	196.6
21	77	81	79.87	**64**	78	**72.00**	196.0
22	76	81	78.27	**65**	85	**72.00**	167.1
23	**63**	67	**65.00**	64	82	70.70	147.6
24	80	80	80.00	**62**	70	**65.30**	143.7
25	71	75	74.73	**65**	78	**72.20**	167.5
26	81	84	82.07	**65**	75	**71.20**	190.8
27	81	81	81.00	**69**	85	**77.10**	151.8
28	88	88	88.00	**69**	85	**74.20**	167.7
29	77	77	77.00	**62**	70	**65.80**	154.6
30	72	75	73.40	**64**	70	**68.00**	158.6
31	69	74	**71.67**	**66**	77	72.30	181.4
32	62	68	65.53	**58**	64	**60.80**	141.7
33	73	75	73.80	**62**	71	**65.00**	145.6
34	73	75	**70.80**	**64**	77	70.90	178.2
35	70	74	71.33	**63**	74	**69.30**	142.0
36	80	88	86.53	**68**	86	**75.80**	153.7
37	73	77	74.73	**67**	73	**70.80**	171.4
38	72	77	73.87	**63**	75	**67.40**	147.9
39	82	82	82.00	**63**	78	**72.60**	169.8
40	69	76	72.60	**61**	73	**68.00**	163.8
41	76	76	**76.00**	**72**	83	76.70	162.2
42	65	68	**67.07**	**63**	79	67.30	161.4
43	72	75	73.60	**66**	80	**71.30**	152.7
44	68	82	76.00	**62**	81	**72.80**	166.8
45	69	81	**73.40**	**65**	82	75.00	167.6
46	78	78	78.00	**64**	78	**70.10**	152.7
47	78	82	78.27	**71**	78	**72.80**	166.3
48	75	77	76.53	**64**	72	**68.50**	167.2
49	73	83	80.07	**72**	78	**74.50**	158.5
50	74	85	76.07	**64**	73	**68.10**	166.6
#Mejores	1		7	49		43	

Los valores en **negrita** son la mejor media de cada instancia y la mejor solución encontrada.

Capítulo 9

Conclusiones

9.1. Aportaciones

A lo largo de este trabajo hemos abordado el problema de la compilación en circuitos cuánticos. Este problema de gran relevancia surge debido a que en las arquitecturas cuánticas actuales se limita la interacción entre qubits, de tal forma que solo se pueden ejecutar puertas lógicas entre qubits adyacentes. Por tanto, se requerirá la introducción de puertas swap, que intercambian la posición de dos qstates. Ademas, el efecto de decoherencia degrada el rendimiento de programas cuánticos según avanza el tiempo, por lo que es esencial diseñar circuitos cuya duración total (makespan) sea lo menor posible.

En particular nos hemos centrado en el modelo descrito en el paper [2]: (i) la clase de circuitos Quantum Approximate Optimization Algorithm (QAOA) que representan algoritmos para resolver el problema MaxCut, y (ii) una arquitectura hardware específica inspirada en la propuesta por Rigetti Computing Inc [7]. Consideramos tres arquitecturas distintas, con 8, 21 y 40 qubits respectivamente.

El problema se puede modelar como un problema de planificación o scheduling, donde los qubits representan recursos y las puertas son las tareas que hay que ejecutar. Hemos propuesto un algoritmo memético, debido a que los híbridos de algoritmo genético y búsqueda local suelen lograr resultados excelentes en problemas de scheduling. Nuestro algoritmo genético utiliza una codificación y unos operadores genéticos específicamente diseñados para el problema en cuestión. Su característica más notable es quizás la optimización del problema en varios pasos, descomponiendo el problema global en subproblemas. La búsqueda local también ha sido específicamente diseñada para este problema, y en particular proponemos varias estructuras de vecindad, que pueden ser eficientemente evaluadas gracias a procedimientos para estimar su calidad en tiempo lineal.

En el estudio experimental hemos analizado nuestra propuesta y la hemos comparado con el estado del arte, obteniendo resultados competitivos, y utilizando además tiempos de ejecución comparables. Creemos que la principal razón de la eficiencia de nuestro enfoque es el equilibrio adecuado entre la diversificación y la intensificación en la búsqueda. La diversificación se logra mediante el algoritmo genético y sus operadores, mientras que la intensificación es provista por una combinación de estructuras de vecindad eficientes. La descomposición del problema permite resolverlo de forma más eficiente, reduciendo de forma efectiva el espacio de búsqueda, ya que

hemos comprobado experimentalmente que la resolución del problema completo es excesivamente compleja.

9.2. Trabajo futuro

Hay varias posibilidades para mejorar el algoritmo propuesto. Podríamos por ejemplo plantearnos el desarrollo de estructuras de vecindad que permitan introducir o eliminar puertas swap innecesarias. La generación heurística de la población inicial también podría ser un método efectivo para llegar a mejores soluciones, debido a que la evolución comenzaría desde un conjunto de individuos de mayor calidad. Otra posibilidad es diseñar un planificador más avanzado, que construya planificaciones mas eficientes a partir de los cromosomas.

Sin embargo, la línea de trabajo futuro posiblemente más interesante es considerar una serie de ampliaciones al problema, inicialmente propuestas en el paper [8], y también consideradas posteriormente en [4]. Estas ampliaciones son: 1) inicialización variable de qstates (QCCP-V) y 2) crosstalk o diafonía (QCCP-X). La inicialización variable de qstates permite asignar los distintos valores iniciales qstate q_i a qubits n_i, en lugar de realizar una inicialización por defecto. Esto dificulta el problema, ya que las posiciones iniciales de los qubits serán parámetros adicionales que deberemos optimizar, pero a cambio es previsible que se podrán obtener soluciones con menor makespan. Por otra parte, la restricción crosstalk prohibe la ejecución concurrente de pares de puertas que comparten los mismos qubits vecinos. El objetivo en este caso es evitar las posibles interferencias que puedan producirse entre los qubits. Previsiblemente, el añadido de esta restricción hará que el makespan empeore.

También hay que tener en cuenta que el problema que hemos tratado es de gran relevancia en la actualidad y la investigación avanza de forma muy rápida. Por tanto, una de las líneas vitales de trabajo futuro será ir adaptando los métodos de resolución a las arquitecturas hardware que se vayan consolidando y a las tecnologías que vayan surgiendo.

Bibliografía

[1] D. Venturelli, M. Do, B. O'Gorman, J. Frank, E. Rieffel, K. E. Booth, T. Nguyen, P. Narayan, S. Nanda, Quantum circuit compilation: An emerging application for automated reasoning, in: S. Bernardini, K. Talamadupula, N. Yorke-Smith (Eds.), Proceedings of the 12th International Scheduling and Planning Applications Workshop (SPARK 2019), 2019, pp. 95–103.

[2] D. Venturelli, M. Do, E. G. Rieffel, J. Frank, Temporal planning for compilation of quantum approximate optimization circuits., in: Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence (IJCAI 2017), 2017, pp. 4440–4446.

[3] A. Oddi, R. Rasconi, Greedy randomized search for scalable compilation of quantum circuits, in: W.-J. van Hoeve (Ed.), CPAIOR 2018: Integration of Constraint Programming, Artificial Intelligence, and Operations Research, Springer International Publishing, Cham, 2018, pp. 446–461.

[4] R. Rasconi, A. Oddi, An innovative genetic algorithm for the quantum circuit compilation problem, in: Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence, Vol. 33, 2019, pp. 7707–7714.

[5] S. Chand, H. K. Singh, T. Ray, M. Ryan, Rollout based heuristics for the quantum circuit compilation problem, in: 2019 IEEE Congress on Evolutionary Computation (CEC), 2019, pp. 974–981. doi:10.1109/CEC.2019.8790000.

[6] E. Farhi, J. Goldstone, S. Gutmann, A quantum approximate optimization algorithm, arXiv preprint arXiv:1411.4028.

[7] E. A. Sete, W. J. Zeng, C. T. Rigetti, A functional architecture for scalable quantum computing, in: 2016 IEEE International Conference on Rebooting Computing (ICRC), IEEE, 2016, pp. 1–6.

[8] K. E. Booth, M. Do, J. C. Beck, E. Rieffel, D. Venturelli, J. Frank, Comparing and integrating constraint programming and temporal planning for quantum circuit compilation, in: Twenty-Eighth International Conference on Automated Planning and Scheduling (ICAPS 2018), 2018, pp. 366–374.

[9] A. Botea, A. Kishimoto, R. Marinescu, On the complexity of quantum circuit compilation, in: Eleventh Annual Symposium on Combinatorial Search (SOCS), 2018, pp. 138–142.

[10] D. C. Mattfeld, Evolutionary Search and the Job Shop Investigations on Genetic Algorithms for Production Scheduling, Springer-Verlag, 1995.

[11] C. Cotta, A. J. Fernàndez, Memetic Algorithms in Planning, Scheduling, and Timetabling, Springer Berlin Heidelberg, Berlin, Heidelberg, 2007, pp. 1–30.

[12] C. R. Vela, R. Varela, M. A. González, Local search and genetic algorithm for the job shop scheduling problem with sequence dependent setup times, Journal of Heuristics 16 (2010) 139–165.

[13] B. Freisleben, P. Merz, A genetic local search algorithm for solving symmetric and asymmetric traveling salesman problems, in: Proceedings of IEEE International Conference on Evolutionary Computation, 1996, pp. 616–621.

[14] R. Varela, C. R. Vela, J. Puente, A. Gómez, A knowledge-based evolutionary strategy for scheduling problems with bottlenecks, European Journal of Operational Research 145 (2003) 57–71.

[15] P. Van Laarhoven, E. Aarts, K. Lenstra, Job shop scheduling by simulated annealing, Operations Research 40 (1992) 113–125.

[16] E. Nowicki, C. Smutnicki, A fast taboo search algorithm for the job shop scheduling problem, Management Science 42 (1996) 797–813.

[17] E. Taillard, Benchmarks for basic scheduling problems, European Journal of Operational Research 64 (1993) 278–285.

[18] Microsoft, Windows 10 Pro, urlhttps://www.microsoft.com/p/windows-10-pro/df77x4d43rkt, accedido el 16-07-2020 (2020).

[19] Microsoft, Visual Studio, urlhttps://www.microsoft.com/es-es/store/collections/visualstudio, accedido el 16-07-2020 (2020).