Automatically Translating Quantum Programs from a Subset of Common Gates to an Adiabatic Representation

Malcolm Regan[1], Brody Eastwood[1], Mahita Nagabhiru[1], and Frank Mueller[1][0000−0002−0258−0294]

Department of Computer Science, North Carolina State University, USA
mueller@cs.ncsu.edu

Abstract. Adiabatic computing with at least 3-local Hamiltonians has been theoretically shown to be equivalent to the gate model of quantum computing. But today's quantum annealers, namely D-Wave's 2000Q platform, only provide a 2-local Ising Hamiltonian abstraction. This raises the question what subset of gate programs can be expressed as quadratic unconstrained binary problems (QUBOs) on the D-Wave. The problem is of interest because gate-based quantum platforms are currently limited to 20 qubits while D-Wave provides 2,000 qubits. However, when transforming entire gate circuits into QUBOs, additional qubits will be required.

The objective of this work is to determine a subset of quantum gates suitable for transformation into 2-local Ising Hamiltonians under a common qubit base representation such that they comprise a compound circuit suitable for pure quantum computation, i.e., without having to switch between classical and quantum computing for different bases. To this end, this work contributes, for the first time, a fully automated method to translate quantum gate circuits comprised of a subset of common gates expressed as an IBM Qiskit program to 2-local Ising Hamiltonians, which are subsequently embedded in the D-Wave 2000Q chimera graph. These gate elements are placed in the chimera graph and augmented by constraints that enforce inter-gate logical relationships, resulting in an annealer embedding that completely characterizes the overall gate circuit. Annealer embeddings for several example quantum gate circuits are then evaluated both on D-Wave 2000Q hardware.

Keywords: Quantum Computation · Quantum Annealing · Quantum Gate Circuits · Adiabatic Computation

1 Introduction

Recent advances in quantum hardware have resulted in the first systems becoming publicly available. On one hand, gate-based quantum computers have been designed, such as the IBM Q, Rigetti's Aspen, or IonQ's systems using using superconducting transmons or ion tubes [2, 11]. On the other hand, quantum

annealing has been promoted by D-Wave's RF-Squids [6]. Both types of systems are available in the cloud and can be programmed using Python, e.g., via IBM's Qiskit in the IBM Q Experience [1], Rigetti's Forest DSK in their Quantum Cloud Services [2], and D-wave's Ocean Software [9] accessible via the cloud through D-Wave Leap [8].

It was shown that adiabatic quantum computing can solve the same problems as gate-based (standard) quantum computing given at least a 3-local Hamiltonian [3, 5, 10]. D-Wave supports a 2-local Ising Hamiltonian in their 2000Q system, which is why it is believed to only solve a subset of the problems that can be expressed by gate-based (general) quantum machines. In fact, D-Wave's programming abstraction is specifically catering to optimization problems while gate-based abstractions map to quantum gates, e.g., by expressing programs as circuits of gates in OpenQASM [7].

In 2014, Warren outlined how a set of universal quantum gates could be realized in adiabatic form using D-Wave's annealing abstraction [12]. This is demonstrated, among others, for C-NOT, Toffoli (CC-NOT), Swap and C-Swap (Fredkin) gates in a $\{0, 1\}$ base of qubit states, and for the Hadamard gate in a two-vector $\{|0\rangle, |1\rangle\}$ base.

In this paper, we contribute a framework to automatically translate gate-based circuits into adiabatic 2-local Hamiltonians expressed as quadratic unconstrained binary optimization problems (QUBOs). We constrain ourselves to a subset of quantum gates in the common $\{0, 1\}$ base so that an entire circuit can be expressed as a single QUBO. This allows us, given a Qiskit program suitable for IBM Q execution, to generate an equivalent Ocean program that can execute on a D-Wave machine. Such a translation is significant since today's gate computers are constrained to 20 qubits for IBM Q (or 19 qubits for Rigetti's available platform), while D-Wave supports around 2,000 qubits on their latest publicly available platform, which enables experimentation at a different scale.

The objectives of this work are (1) to identify a subset of gates suitable for translation, (2) to demonstrate the feasibility of auto-translating entire circuits of these quantum gates to adiabatic programs, (3) to assess the cost of ancilla qubits required to express gates in QUBOs, (4) to find an embedding into D-Wave's Chimera graph for a circuit and assess its cost in extra qubits and circuit lines / wires, and (5) to compare hardware experimentation results with the expected ground state to determine the annealer's ability identify coherent solutions for circuit embedding. We contribute an automated method for encoding quantum gate circuits comprising X, C-NOT, Toffoli, Swap and C-Swap (Fredkin) gates as 2-local Ising Hamiltonians (QUBOs) and embed the resulting representation in the D-Wave 2000Q chimera graph. We provide the 2-local Ising QUBOs with $K_{4,4}$ connectivity, a compete bipartite graph with 8 vertices corresponding to D-Wave's unit cell, for which ground state configurations logically characterize quantum X, C-NOT, Toffoli, and Swap gates. Notice that we do not provide a translation for the Hadamard gate, H, as it requires a different base of qubit states than the above, i.e., one cannot directly embed H in the same circuit. Instead, one would have to transition between quantum and classical

programs, which collapses the quantum state and thus defeats the purpose of quantum computing in first place. These gates supported by our translation constitute building blocks that are placed in the chimera graph and augmented by constraints that enforce inter-gate logical relationships. The resulting annealer embedding is equivalent to the corresponding gate circuit in terms of its computational functionality. In experimental results, we evaluate annealer embeddings for several sample quantum gate circuits on D-Wave hardware.

2 Design and Implementation

In adiabatic computing, the comprehensive state of qubits is annealed via a combination of tunneling and entanglement toward a ground (energy) state. There may be more than one such state, and tunneling aids in not getting stuck in local minima but rather find other ground states, subject to practical considerations of adiabatic computing, such as experienced by near absolute zero Kelvin operation and hardware-induced errors in any practical quantum devices. To this end, D-Wave supports a 2-local Ising Hamiltonian

$$\mathcal{H}(t) = -\sum_{i=0}^{N-2}\sum_{j=0}^{N-1} J_{i,j}\sigma_i\sigma_j - \sum_{i=0}^{N-1} S_i\sigma_i - \Gamma(t)\sum_{i=0}^{N-1}\sigma_i$$

with N qubits $\sigma_i \in \{-1, 1\}$ as vertices, coupler strengths $J_{ij} \in \{-2, 2\}$ that connect σ_i, σ_j and biases (weights) S_i per qubit such that the amplitude, $\Gamma(t)$, of the third term, the traverse field is gradually decreased to drive the aggregate of the first and second term into a ground state, $\mathcal{H}_0$.

A 2-local Hamiltonian is expressed as quadratic unconstrained binary optimization problem (QUBO) that describes a ground state and is subsequently mapped onto D-Wave's 2000Q embedding of qubits respecting the connectivity of qubit pairs. Specifically, D-Wave's inner cell is a $K_{4,4}$ bipartite graph to which we map quantum gates. This embedding of a gate is described in Section 2.1.

The $K_{4,4}$ unit cells are arranged in a 2-dimensional 16×16 grid in a Chimera graph with sparse horizontal and vertical couplings between equivalent qubits of neighboring unit cells. The Chimera graph provides the means to connect unit cells representing a quantum gate with each other to create the desired quantum circuit of a given gate-based quantum program, which is described in Section 2.2.

We then develop an automatic transition from Qiskit programs representing circuits of quantum gates to an equivalent adiabatic representation in a systematic manner in Section 2.3. This translator leverages the class and file structure of IBM's open-source Qiskit API for definitions of quantum gate circuits due to its familiarity and ease-of-use. Specifically, a Qiskit translator was created so that any Qiskit script defining a quantum gate circuit could be used to generate and run a corresponding annealer embedding.

2.1 2-Local Ising Hamiltonians for $K_{4,4}$ Embeddings of Quantum Gates

Gate embeddings for a $\sigma_i \in \{0, 1\}$ base, depicted in Figure 4, were designed to have a ground state characterizing the corresponding quantum gate's logical function. The process by which the gate embeddings used in this project were determined is described below in terms of the C-NOT gate as an example.

The C-NOT gate operates on 2 qubits, a control qubit and a target qubit. Because the target qubit is potentially altered by the C-NOT operation, its value after the C-NOT operation must be considered distinctly. Accordingly, the 8 possible configurations of 3 binary variables — q_0, q_1, and q_2 — are shown in the truth table on the left of Figure 1. Arbitrarily, these variables are designated to represent the control qubit value, the value of the target qubit before the C-NOT operation, and the value of the target qubit after the C-NOT operation, respectively.

ctl q_0	targ q_1	out q_2	
0	0	0	$0 = G$
0	0	1	$S_2 > G$
0	1	0	$S_1 > G$
0	1	1	$S_1 + S_2 + J_{12} = G$
1	0	0	$S_0 > G$
1	0	1	$S_0 + S_2 + J_{02} = G$
1	1	0	$S_0 + S_1 + J_{01} = G$
1	1	1	$S_0 + S_1 + S_2 + J_{01} + J_{02} + J_{12} > G$

Fig. 1: Truth table showing all possible logical combinations of 3 binary variables and the corresponding Ising Hamiltonian constraints for a C-NOT operation. Ground state configurations are highlighted in green.

Of the 8 possible configurations, only 4 correspond to a qubit transformation performed by a C-NOT gate. As such, it is these configurations that we require to correspond to the lowest energy of the Ising Hamiltonian. This results in a set of constraints — one for each row of the truth table shown in Figure 1 — in terms of 2-local Ising Hamiltonian variables, S_i and J_{ij}, and ground state energy, G. These constraints are shown on the right of Figure 1. S_i and J_{ij} are referred to as qubit biases and coupler strengths, respectively.

These inequalities were then solved under the constraint that the solution comprised only integer values between -10 and 10. (Notice that this range is later mapped to some $S_i \in \{-2, 2\}$ to meet the D-Wave embedding constraints.) If a given set of constraints had no solution, as in the case of the C-NOT, an ancilla variable was added as shown in Figure 2 and a system of constraints was again generated and a solution was sought.

The graph of the resulting C-NOT Ising Hamiltonian is shown on the left of Figure 3. This graph however, is not compatible D-Wave's chimera graph. Recall that the chimera graph is a 16 by 16 array of $K_{4,4}$ unit cells whose right-hand

ctl q_0	targ q_1	out q_2	anc q_a	
0	0	0	0	$0 = G$
0	0	0	1	$S_a > G$
0	0	1	0	$S_2 > G$
0	0	1	1	$S_2 + S_a + J_{2a} > G$
0	1	0	0	$S_1 > G$
0	1	0	1	$S_1 + S_a + J_{1a} > G$
0	1	1	0	$S_1 + S_2 + J_{12} = G$
0	1	1	1	$S_1 + S_2 + S_a + J_{12} + J_{1a} + J_{2a} > G$
1	0	0	0	$S_0 > G$
1	0	0	1	$S_0 + S_a + J_{0a} > G$
1	0	1	0	$S_0 + S_2 + J_{02} = G$
1	0	1	1	$S_0 + S_2 + S_a + J_{02}J_{0a} + J_{2a} > G$
1	1	0	0	$S_0 + S_1 + J_{01} > G$
1	1	0	1	$S_0 + S_1 + S_a + J_{01} + J_{0a} + J_{1a} = G$
1	1	1	0	$S_0 + S_1 + S_2 + J_{01} + J_{02} + J_{12} > G$
1	1	1	1	$S_0 + S_1 + S_2 + S_a + J_{01} + J_{02} + J_{12} + J_{0a} + J_{1a} + J_{2a} > G$

Fig. 2: Truth table showing all possible logical combinations of binary variables after an ancilla variable is added and the corresponding Ising Hamiltonian constraints for a C-NOT operation. Ground state configurations are highlighted in green.

nodes are connected horizontally and left-hand nodes are connected vertically. Therefore, the graphs obtained by solving the system of constraints were modified into logically equivalent graphs conforming to the $K_{4,4}$ connectivity of a unit cell. This was done by splitting qubits requiring connections having no corresponding coupler in the chimera graph. For example, as shown on the left of Figure 3, the vertical connections, J_{1a} and J_{02}, have no physical counterpart in the chimera graph. To remedy this, one of the logical qubits being coupled through these connectors can be represented by two physical qubits, one on each side of the graph, rendering the all the connections physically realizable. On the right of Figure 3 is the graph that results from splitting q_a into q_a and $q_{a'}$ and q_2 into q_2 and $q_{2'}$. Qubit biases for these new qubits are increased from their original value by a positive offset, δ, and the coupling strength between them is set to the negative of the bias of the original qubit minus 2δ. This ensures that, when both physical qubits are in sync, the embedding is equivalent to the corresponding configuration in the unmodified graph. For example, referring to the graph on the right of Figure 3, when both q_a and $q_{a'}$ are equal to 1, an energy of

$$(S_a + \delta) + (S_a + \delta) + (-S_a - 2\delta) = S_a$$

is contributed to the system. This is the same energy contribution made to the system represented by the graph on the left of Figure 3 when q_a is equal to 1. Further, splitting qubits in this way ensures that the energy of any new logical configurations introduced by the new qubits are above the ground state by at least δ. In this work, δ was set to 5.

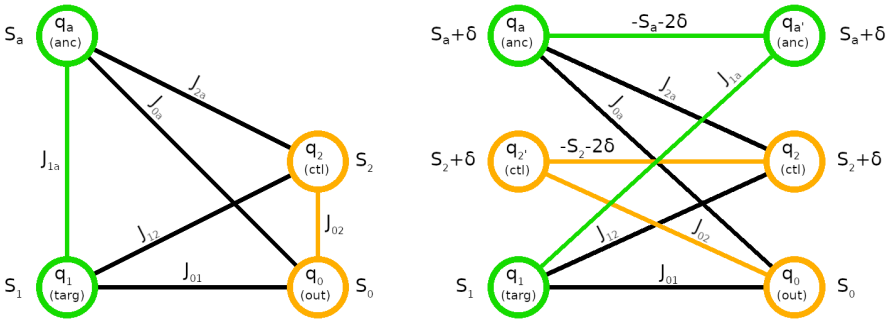
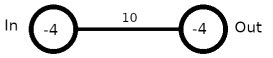


Fig. 3: (Left) Ising Hamiltonian graph obtained from system of constraints. Sections highlighted in green and orange are not compatible with D-Wave's chimera graph. (Right) Logically equivalent graph modified to conform to chimera graph unit cell connectivity. Sections highlighted in green and orange indicate the modifications to the graph.

Finally, to ease the process of embedding the overall gate circuit (described in Section 2.2), we also required that any qubit representing a gate input be on one side of the graph and any qubit that could be used as an input to a later gate be on the opposite side of the graph. For example, the Toffoli gate pictured in Figure 4(c) has 3 inputs (Target, Control 1, and Control 2) and 3 outputs (Out, Control 1, and Control 2). The value of the Control qubits are not transformed by the gate and as such must be present on both sides of the graph. This ensures that the most recent state of these qubits are easily accessible to other, later gates. The value of the Target qubit is transformed by the gate into the value of the Out qubit. As such, the Target qubit is required to be present on the input side of the gate embedding and the Out qubit is required to be on the output side of the gate embedding. In the Toffoli embedding, the Out qubit also happens to be present on the input side, but this is for the purpose of making the connections needed to form a valid gate embedding.

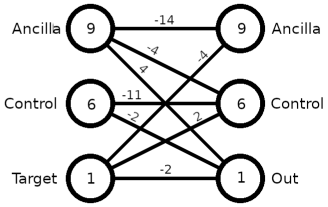
The above process was carried out to determine embeddings for X, Toffoli, and Swap gates. A 2-local Ising Hamiltonian assuming full graph connectivity for the C-NOT gate was sufficiently determined in a previous work [12]. This Ising Hamiltonian was used as a starting point and modified to conform to the criteria described above. A 2-local Ising Hamiltonian assuming full graph connectivity for the C-Swap function was also determined in this work. However, the graph of this Hamiltonian could not be modified into a form meeting the criteria described above. Specifically, configuring the connections comprising the full connectivity graph to conform to chimera graph connectivity requires at least 2 unit cells. Also, satisfying our requirement that gate inputs and outputs be present on certain sides of a cell requires the use of more resources, and further removes the symmetries that the circuit embedding algorithm relies on. Due to these

problems, the C-Swap gate was implemented with 2 C-NOT embeddings and a Toffoli embedding, connected as illustrated on the right of Figure 5.



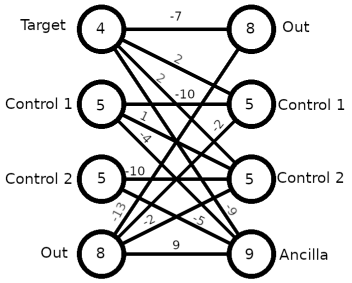
In	Out
0	1
1	0

(a)



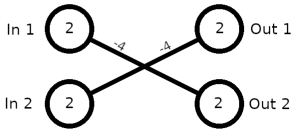
Control	Target	Out
0	0	0
0	1	1
1	0	1
1	1	0

(b)



Control 2	Control 1	Target	Out
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

(c)



In 2	In 1	Out 2	Out 1
0	0	0	0
0	1	1	0
1	0	0	1
1	1	1	1

(d)

Fig. 4: (Left) Embeddings for (a) X, (b) C-NOT, (c) Toffoli, and (d) Swap gates illustrated to show qubit names, qubit biases, and coupler strengths. (Right) Ground state configurations of logical variables.

Fig. 5: (Left) C-Swap gate symbol with qubit names indicated (Right) Functionally equivalent circuit used to implement C-Swap gate.

2.2 Embedding the Problem in the Chimera Graph

Determining an optimal embedding for an arbitrary annealer problem is NP-complete and the heuristics commonly used to determine working embeddings can often fail when the problem is complex. We implement a process that exploits properties of quantum gate circuits and the symmetries of the gate embeddings determined above to reliably construct working embeddings for gate circuits.

Chaining Logical connections between gate embeddings are made via "chains" of qubits that encode a single logical qubit. A chain is created by assigning the biases of the qubits that comprise the chain and the strengths of the couplers between them similarly to how they were assigned when splitting a logical qubit (Figure 3). Specifically, each section of the chain is constructed by offsetting the biases of the qubits being connected by δ and assigning to the corresponding coupler a strength of -2δ. This is done one section at a time, from the earlier gate to the later gate. An example of the resulting chain is shown in Figure 6 for the case of two X gates connected into an identity function.

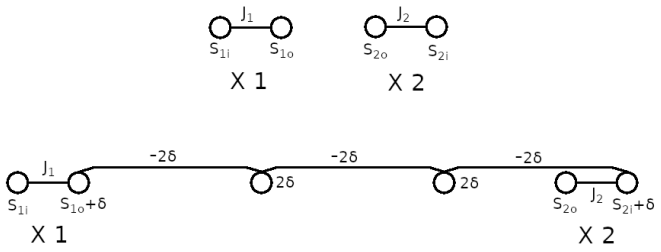


Fig. 6: Chaining example. (Top) Illustration of gate elements X1 and X2 showing names of qubit biases and coupler strengths. Qubits whose biases are denoted with the letter "i" are input qubits and those denoted with the letter "o" are output qubits. (Bottom) An embedding of the identity function resulting from chaining the output of X1 to the input of X2.

Chimera Graph Cell Designations The "signal flow" of a gate circuit naturally lends itself as an organizing principle for the problem embedding. To best translate the notion of a signal flow to the Chimera graph, gate embeddings are placed as shown in Figure 7. The connectivity between adjacent gate embeddings is highlighted and cell designations are indicated. Each gate embedding has corresponding sections of the graph designated for delivering chains to non-adjacent gates (chain output column) and assembling its inputs (input assembly cell) denoted in Figure 7 as CO and IA, respectively.

Gate embeddings, designed to have inputs and outputs on either side of a single bipartite cell, are reflected depending on whether the gate number is even or odd. Specifically, if the gate number is even, the output column of the gate

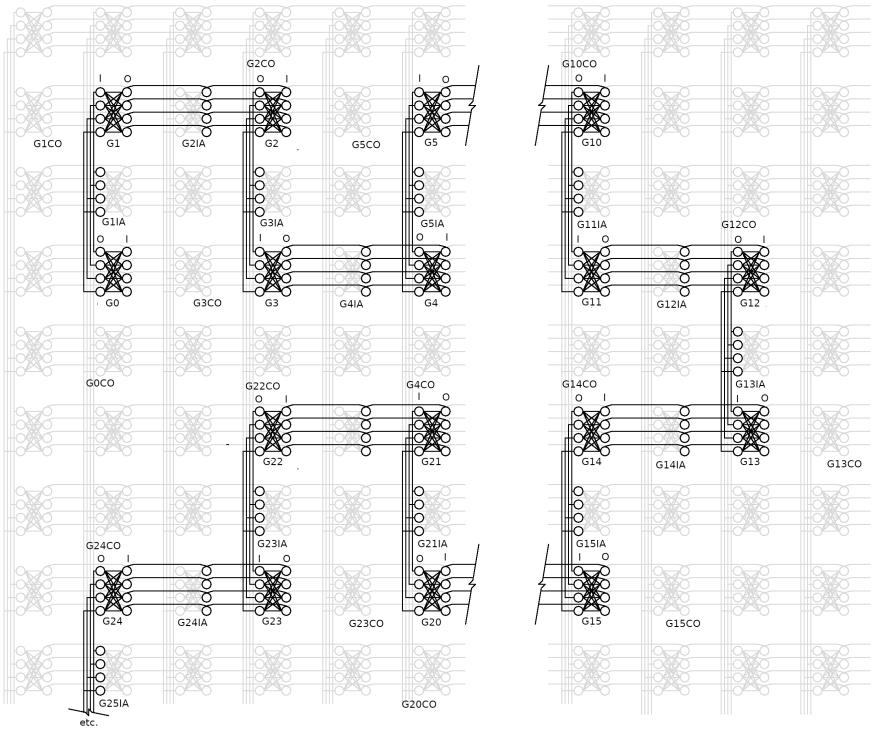


Fig. 7: Chimera graph cell designations for gate circuit embeddings. G1, G2, etc. = Gate 1, Gate 2, etc.; IA = Input assembly cell; CO = Chain out column; I/O = Input/Output column

embedding is on the left and the input column is on the right. If the gate number is odd, the input column is on the left and the output column is on the right.

If an input of a given gate is dependent on an output of a previous adjacent gate, the rows of the new gate embedding are permuted to align its input with the previous output and a chain is made through its input assembly cell. Rows of unit cells can be permuted without change to the network topology of the gate embedding, which makes connections between adjacent gates trivial.

If an input of a gate is dependent on an output of a previous non-adjacent gate, a chain is routed from the chain output column of the earlier gate through empty positions on the Chimera graph to the input assembly cell of the new gate. The role of chain out column and input assembly cell designations in chain routing are illustrated in Figure 8. New gates first align its rows with connections to adjacent gates, then non-adjacent gates. Rows with no dependencies are assigned position in the bipartite cell last.

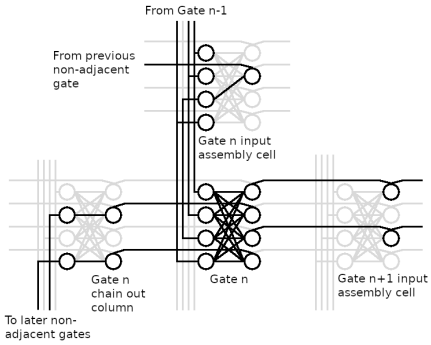


Fig. 8: Detail illustrating role of input assembly cells and chain out columns in making inter-gate connections

2.3 Implementing the Qiskit to D-Wave Ocean Translator

Qiskit is an open-source Python API developed for the implementation and execution (or simulation) of quantum gate circuits on IBM quantum computers [1]. As this API provides a convenient and intuitive framework with which quantum gate circuits can be defined, this translator project was built within its class structure. This was achieved with the objective that any Qiskit script defining a quantum gate circuit could be used to generate and run a corresponding annealer embedding. Our approach allows a gate circuit in Qiskit to be executed (a) on IBM Q quantum hardware, (b) in simulation using IBM's APIs, or, after translation, (c) on D-Wave's quantum annealer hardware. Given IBM Q's constraint to at most 20 qubits at this time, Qiskit programs requiring more than 20 qubits, which may be very slow in simulation, can be executed on D-Wave hardware in a fraction of the corresponding simulation time.

Our translator is implemented as new backend to the Qiskit source code in terms of the `AnnealerGraph` class, whose methods handle the configuration, placement, and chaining together of gate embeddings. An instance of `AnnealerGraph` was added as an attribute to Qiskit's `QuantumCircuit` class, which is a central object in the Qiskit framework, whose attributes are operated on or used by every Qiskit function relevant to this project.

In a Qiskit script, an instance of `QuantumCircuit` is initialized as a collection of `QuantumRegister` and `ClassicalRegister` instances. A gate circuit is then defined via `QuantumCircuit` methods that operate on `QuantumRegister` objects.

Our translator thus implements a modified version of Qiskit, where an `AnnealerGraph` instance is initialized in the `QuantumCircuit` initialization function and builds data structures needed to construct an annealer embedding from Qiskit instructions. `AnnealerGraph` has a dictionary attribute, `qubits`, in which each qubit in the gate circuit, assigned a name in the initialization function of `QuantumCircuit`, has a corresponding entry (with keys corresponding to Qiskit register names). This dictionary keeps track of which annealer graph nodes are assigned to a given logical qubit, in what order these nodes were assigned, and

whether or not the final state of this qubit is considered an output (i.e., whether the `measure` function was used on this qubit). There is also an entry in `qubits` that keeps track of annealer graph nodes that do not correspond to logical qubits.

Annealer graph nodes, identified by D-Wave Ocean as numbers, are added to the lists comprising the `qubits` dictionary as gates and chains between gates are added to the graph. Dictionary objects used as the input arguments to the D-Wave Ocean embedding compilation function, `qubitbiases` and `couplerstrengths`, are also built as gates and chains are assigned to the graph. The `qubitbiases` dictionary contains as keys a number identifying a given node in the chimera graph. The value associated with a given key is the bias itself. The `couplerstrengths` dictionary contains as keys a tuple identifying the two qubits being coupled (smaller number first). The value associated with a given key is the coupler strength.

`AnnealerGraph` contains methods for adding the circuit's gate elements (`addX`, `addCNOT`, etc.). Therefore, Qiskit functions for adding gates to a quantum circuit were modified to call the appropriate `AnnealerGraph` method in lieu of the original Qiskit code. In general, `AnnealerGraph` methods for adding gate elements to the circuit embedding are structured as follows. `AnnealerGraph` has as an attribute a counter that indicates how many gates have already been placed in the circuit embedding. This is used to determine where in the graph the new gate is placed per the cell designations described in Section 2.2. Next, connections to previously placed gate embeddings are determined. For each input in the gate, the last element in the `qubit` dictionary entry for the corresponding gate circuit qubit indicates the most recent state of that of that qubit and its position. If its position is in an adjacent gate, the row containing the corresponding input of the new gate is placed in the unit cell to align it with its connection in the previous gate. These qubits are then connected with a chain through the new gate embedding's input assembly cell. If the new gate requires a connection from a non-adjacent gate, a chain is made from the last instance of the qubit to the input assembly cell of the new gate. The input of the new gate is then assigned a position in the gate cell aligned with the position of its connection. Once gate qubits with dependencies are placed in the gate cell, qubits with no dependencies are placed in remaining positions.

The last significant modification to Qiskit was to its `execute` function. The Qiskit `execute` function was modified to make final adjustments to the circuit embedding, execute the embedding on D-Wave hardware, and report the results. In our code, when `execute` is called, the user is prompted, for each qubit in the gate circuit, to answer whether the initial state of the qubit should be constrained to a value of zero. If the user answers that it should be and it was not earlier identified as a circuit output by the `measure` command, it is assumed this qubit is an ancilla and as such is not reported in the results. If the user answers that it should be constrained to zero and it has been identified as a circuit output by the `measure` command, the output values are still reported, but the input values are not. The initial state of a logical qubit is constrained to a value of zero by adding 5 to the bias of the first physical qubit associated with it. Results are

reported with input variable values on the left and output variable values on the right.

3 Experimental Results

3.1 Circuit for Comparison of 1-Qubit Numbers

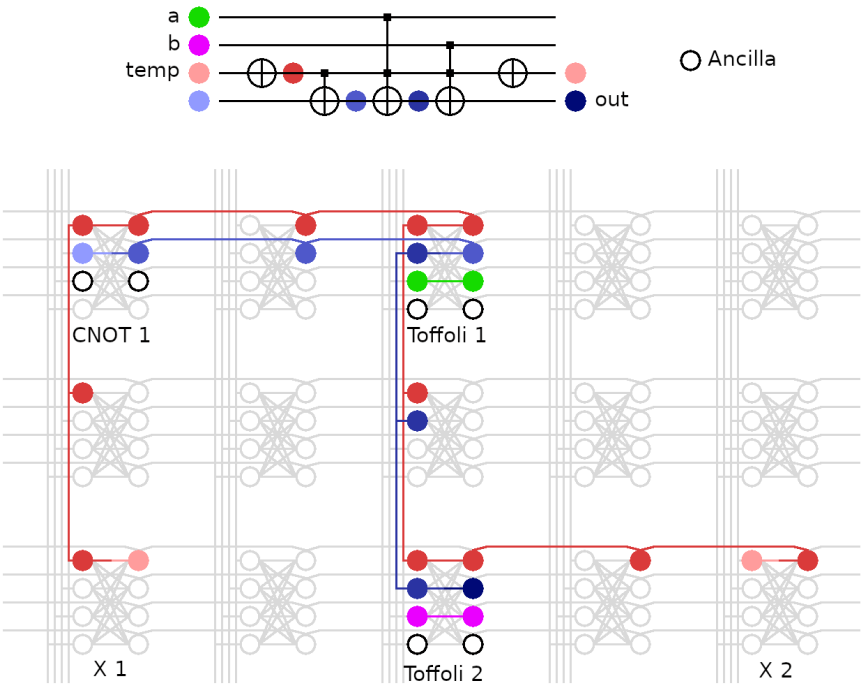


Fig. 9: Embedding of equivalence check circuit for 1-qubit numbers. Colors indicate logical qubits.

Shown at the top of Figure 9 is a quantum circuit whose output is $|1\rangle$ if its two input qubits, $|a\rangle$ and $|b\rangle$, are in the same logical state and $|0\rangle$ if they are not. The temporary qubit is not necessary for a 1-qubit equivalence circuit such as this but temporary registers are needed for similar circuits when comparing multi-bit inputs. The temporary register is included here to make this example more interesting. The main illustration in Figure 9 shows the embedding automatically generated from a Qiskit program that defines the circuit depicted. This embedding anneals as expected. If the initial states of the output and temporary qubits are constrained to be zero there are 4 valid results. All 4 results are reliably obtained within 100 samples. The embedding uses 32 physical qubits and 48 couplers, 12 of which are used for inter-gate connections. This embedding is clearly not optimal. An optimal graph for a circuit of this size and functionality is

easily obtained using the process by which the gate embeddings were determined (Section 2.1). An optimal graph for this circuit's function (XNOR) is shown in Figure 10. It uses 6 qubits and 8 couplers and is contained within a single $K_{4,4}$ cell. Therefore, the generated circuit uses about 6 times more resources than is optimal.

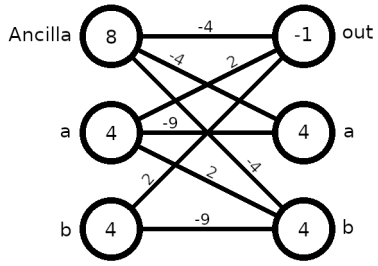


Fig. 10: An optimal XNOR graph with matching functionally to the generated 1-qubit equivalence circuit embedding

3.2 Circuit for Comparison of 5-Qubit Numbers

The 1-qubit equivalence circuit was expanded into a circuit for comparison of 5-qubit numbers. The current chain routing algorithm caused the embedding for this circuit to become congested when routing temporary qubits from earlier in the circuit to be uncomputed at the end of the circuit, which resulted in a graph that did not map to the chimera graph. Specifically, the current chain routing algorithm does not yet implement any precautions against routing chains into graph positions from which there are no further available connections. When this occurs, the algorithm is forced to assign a connection via a nonexistent coupler. Due to this, gates used for uncomputing the temporary qubits were not included in the circuit, and without them, the resulting embedding conformed to chimera graph connectivity and was able to be annealed on D-Wave 2000Q hardware.

The embedding generated for the 5-qubit equivalence circuit uses 156 physical qubits and 240 couplers, 68 of which are used in chains between gates. These values are also about 5 times larger than those of the 1-qubit equivalence circuit, as expected. However, this embedding does not anneal as effectively as the 1-qubit equivalence circuit. Only about 20 of the 1,024 valid results are obtained within 10,000 samples.

3.3 Adder for 1-Qubit Numbers

The top of Figure 11 shows a quantum circuit implementing a full adder function. The main illustration in this figure shows the embedding generated from a Qiskit script defining this circuit. This embedding is composed of 79 physical qubits and 110 couplers, 47 of which are used for inter-gate connections. Several improvements that could be made to this embedding are apparent. Most obviously, the chains connecting gates could be routed more efficiently. An optimal

full adder annealer embedding uses 8 qubits and 13 couplers and fits within a single $K_{4,4}$ cell [4]. So, in terms of bipartite cell embeddings used, the generated full adder embedding is 6 times larger than the optimal case. Considering resources used to connect gates, the generated embedding is about 11 times larger.

The generated full adder embedding anneals as expected. There are 8 valid results if the initial states of sum and carry-out qubits are constrained to be zero. All 8 results are reliably obtained within 400 samples.

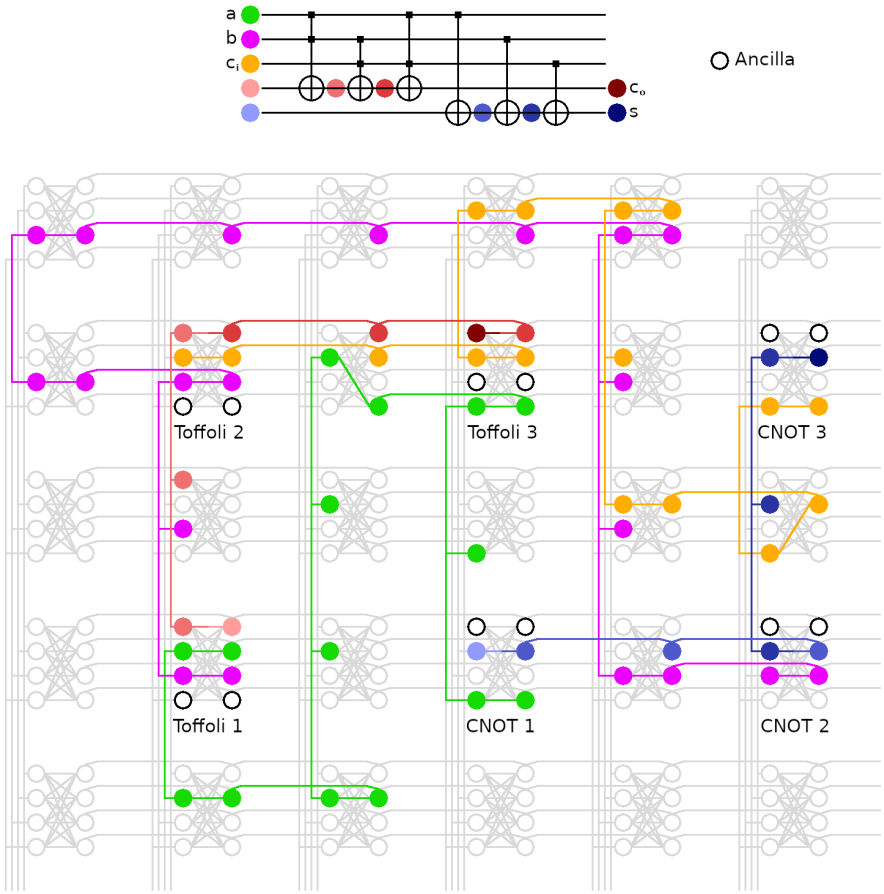


Fig. 11: Embedding of adder for 1-qubit numbers. Colors indicate logical qubits.

3.4 Adder for 4-Qubit Numbers

The 1-qubit adder was expanded to implement a 4-qubit adder. The corresponding embedding used 322 physical qubits and 430 couplers, 212 of which were used for inter-gate connections. Qubits in gate cells G9, G10, and G17 were down on the D-Wave machine, and as such these cells were not used. Due to having to route around these cells, extra qubits and couplers were included in the em-

bedding. Nonetheless, the number of qubits and couplers used in the generated embedding are still approximately 4 times that of the 1-qubit adder embedding.

This embedding does not anneal as effectively as the 1-qubit adder. There are 256 valid ground states when the initial state of the output qubits are constrained to zero. Only 16 of these are ground states are found within 10,000 samples.

3.5 Multiplication Circuit for 2-Qubit Numbers

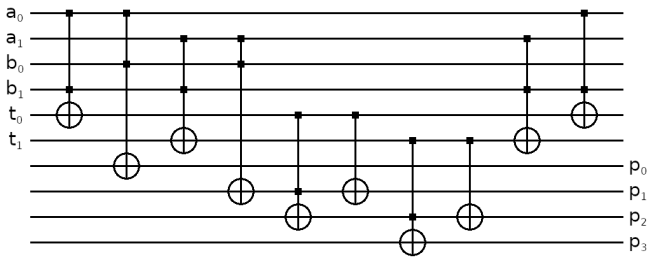


Fig. 12: 2-qubit multiplication circuit

The quantum circuit pictured in Figure 12 takes 2-qubit numbers, a_1a_0 and b_1b_0, as input and computes their product $p_3p_2p_1p_0$. The embedding generated from a Qiskit script defining this quantum circuit comprises 200 physical qubits and 262 couplers, 142 of which were used for inter-gate connections. In related work, an embedding for a 3-bit multiplication function was implemented by making appropriate connections between single cell embeddings for full adder, half adder, and AND functions [4]. An embedding for a 2-bit multiplication function constructed in this way would comprise 4 AND embeddings and 2 half adder embeddings, which require 28 qubits and 32 couplers between them. Assuming a similar chaining scheme, about 16 qubits and 23 couplers would be required to connect the minor embeddings. Therefore, the embedding would require approximately 44 qubits and 55 couplers. Compared to this embedding, the embedding generated here uses about 6 times the amount of qubits, and about 3 times the number of couplers. Note that, as in the case of the 4-qubit adder, qubits in gate cells G9, G10, and G17 were down on the D-Wave machine, and so these cells were not used. This resulted in extra qubits and couplers being included in the generated embedding.

This embedding anneals as expected. If the initial state of output qubits and temporary qubits, t_0 and t_1, are constrained to zero there are 16 valid results. All 16 results have been obtained in 10,000 samples.

4 Conclusion

We contributed an automatic translation scheme from a set of quantum gates, expressed as a Qiskit circuit suitable for execution on the IBM Q platform, to an adiabatic circuit with an equivalent 2-local Ising Hamiltonian that is embedded

on a chimera graph and expressed as an Ocean program suitable for D-Wave 2000Q execution. Experiments indicated that the generated target circuits were using six times more qubits and three times more couplers that the source circuits. In future work, we plan to develop optimization techniques to reduce the number of resources required by exploiting inter-gate embeddings within unused couplers of a cell representing a gate and by reducing qubits by fusing gates together. We also intend to further extend the set of gates suitable for adiabatic transformation in circuits.

References

1. Ibm qiskit (2018), https://github.com/Qiskit/qiskit-terra
2. Rigetti forest (2018), https://www.rigetti.com/forest
3. Aharonov, D., van Dam, W., Kempe, J., Landau, Z., Lloyd, S., Regev, O.: Adiabatic quantum computation is equivalent to standard quantum computation. SIAM J. Comput. **37**(1), 166–194 (Apr 2007). https://doi.org/10.1137/S0097539705447323, http://dx.doi.org/10.1137/S0097539705447323
4. Andriyash, E., Bian, Z., Chudak, F., Drew-Brook, M., King, A.D., Macready, W.G., Roy, A.: Boosting integer factoring performance via quantum annealing offsets. Tech. rep. (2016)
5. Bacon, D., Flammia, S.T., Crosswhite, G.M.: Adiabatic quantum transistors (2012). https://doi.org/10.1103/PhysRevX.3.021015
6. Boixo, S., Albash, T., Spedalieri, F.M., Chancellor, N., Lidar, D.A.: Experimental signature of programmable quantum annealing. arXiv:1212.1739 (2012), http://arxiv.org/abs/1212.1739
7. Cross, A.W., Bishop, L.S., Smolin, J.A., Gambetta, J.M.: Open quantum assembly language. arXiv:1707.03429 (2017), http://arxiv.org/abs/1707.03429
8. D-Wave: D-wave leap. https://www.dwavesys.com/take-leap
9. D-Wave: D-wave's ocean software. https://ocean.dwavesys.com/
10. Dorit Aharonov, Wim van Dam, J.K.Z.L.S.L.O.R.: Adiabatic quantum computation is equivalent to standard quantum computation. ArXiv e-prints (May 2004), https://arxiv.org/abs/quant-ph/0405098
11. IBM: IBM Q Experience. https://quantumexperience.ng.bluemix.net/qx
12. Warren, R.H.: Gates for adiabatic quantum computing. ArXiv e-prints (Aug 2014), https://arxiv.org/abs/1405.2354

5 Appendix: Sample Qiskit Program

The sample Qiskit program below defines the 1-qubit adder circuit described in Section 3.1. Note that the imports have been modified to indicate the use of the gate-circuit-to-annealer embedding translator (i.e., `'converter.qiskit'` replaces `'qiskit'`). This is the only modification that is required to run a Qiskit program with our translator code.

```
from converter.qiskit import QuantumRegister, ClassicalRegister
from converter.qiskit import QuantumCircuit, execute

# Input Registers: a = qi[0]; b = qi[1]; ci = qi[2]
qi = QuantumRegister(3)
ci = ClassicalRegister(3)

# Output Registers: s = qo[0]; co = qo[1]
qo = QuantumRegister(2)
co = ClassicalRegister(2)

circuit = QuantumCircuit(qi,qo,ci,co)

# Define adder circuit
for idx in range(3):
   circuit.ccx(qi[idx], qi[(idx+1)%3], qo[1])
for idx in range(3):
   circuit.cx(qi[idx], qo[0])
circuit.measure(qo, co)

# Run
execute(circuit)
```

In this program, the `QuantumRegister` and `ClassicalRegister` classes are identical to those used by a regular Qiskit program. The initialization of the `QuantumCircuit` object following the register initializations is identical to that of a regular Qiskit program except that an instance of the `AnnealerGraph` class is initialized within it as an attribute.

The Toffoli and C-NOT gate methods (`ccx` and `cx`, respectively) build the `qubitbiases` and `couplerstrengths` dictionaries that define the embedding and are used as arguments to the D-Wave Ocean compilation method called in the execute method. Truncated versions of the `qubitbiases` and `couplerstrengths` dictionaries constructed by the Qiskit program above are shown below. The keys of these dictionaries indicate a given qubit or coupler between qubits, and the entries indicate the bias of the qubit or strength of the coupler.

```
qubitbiases = {397: 5,    couplerstrengths = {(393, 397): -10,
               393: 10,                       (394, 398): -10,
```

```
398: 5,                          (392, 399): -13,
394: 10,                         (395, 396): -9,
396: 9,                          (395, 397): -4,
399: 8,                          (395, 398): -5,
392: 13,                         (395, 399): 9,
395: 9,                          (392, 396): -7,
  ...                                ...
293: 10,                         (168, 172): -14,
301: 10,                         (168, 175): -4,
299: 10,                         (169, 172): -4,
171: 11,                         (168, 173): 4,
175: 6,                          (171, 175): -11,
173: 1,                          (171, 173): -2,
168: 9,                          (169, 175): 2,
172: 9}                          (169, 173): -2}
```

The measure method is used as an indication that a given qubit register is considered a circuit output, which aids in organizing the results. The `execute` method makes final modifications to the embedding definition given by the `qubitbiases` and `couplerstrengths` dictionaries, runs the embedding on D-Wave 2000Q hardware and reports the result. When the `execute` method is called, the user is prompted to answer whether or not initial values of qubits should be constrained to zero. Qubits are identified using Qiskit's naming scheme in the program and by the order with which they appeared in the initialization of the `QuantumCircuit`. In the case of the 1-qubit adder above, the user would like for the initial state of the sum and carry-out qubits be constrained to zero, and so responds to the prompts from execute as follows:

```
Constrain input of measured qubit q1_0 to be 0 (y/n)? y
Constrain input of measured qubit q1_1 to be 0 (y/n)? y
Constrain input of unmeasured qubit q0_0 to be 0 (y/n)? n
Constrain input of unmeasured qubit q0_1 to be 0 (y/n)? n
Constrain input of unmeasured qubit q0_2 to be 0 (y/n)? n
```

The initial state of a qubit is constrained to zero by adding a positive offset to the bias of the first physical annealer qubit associated with it.

Next, the user is prompted to specify the number of anneals that they would like to run. For the 1-qubit adder, 400 has been shown to sufficient. After this, the embedding is constructed using the D-Wave `BinaryQuadraticModel` method and annealed via the `sample` method

```
bqm = dimod.BinaryQuadraticModel(qb, cs, 0, dimod.BINARY)
response = sampler.sample(bqm, **kwargs)
```

where `qb` and `cs` are copies of the `qubitbiases` and `couplerstrengths` dictionaries, respectively, and `kwargs` contains annealing parameters.

The results of annealing the embedding generated by the Qiskit program are then reported. The results of the embedding generated by the 1-qubit adder program above are shown below:

```
[0, 0, 0, 0, 0]
[0, 0, 1, 1, 0]
[0, 1, 0, 1, 0]
[0, 1, 1, 0, 1]
[1, 0, 0, 1, 0]
[1, 0, 1, 0, 1]
[1, 1, 0, 0, 1]
[1, 1, 1, 1, 1]
```

Results are presented with inputs on the left and outputs on the right, in the order that they were listed when `QuantumCircuit` was initialized. The columns of the results then, from left to right, correspond to qubits `a`, `b`, `ci`, `s`, and `co`. Note that if the initial state of an output is not constrained to zero by the user, its initial state is reported as an input.