# A Hardware-Aware Heuristic for the Qubit Mapping Problem in the NISQ Era

Siyuan Niu, Adrien Suau, Gabriel Staffelbach, Aida Todri-Sanial

# A Hardware-Aware Heuristic for the Qubit Mapping Problem in the NISQ Era

Siyuan NIU
LIRMM, Univ Montpellier
Montpellier, France
siyuan.niu@lirmm.fr

Adrien Suau
LIRMM, Univ Montpellier
Montpellier, France
CERFACS
Toulouse, France
adrien.suau@cerfacs.fr

Gabriel Staffelbach
CERFACS
Toulouse, France
gabriel.staffelbach@cerfacs.fr

Aida Todri-Sanial
LIRMM, CNRS
Montpellier, France
aida.todri@lirmm.fr

## ABSTRACT

Due to several physical limitations in the realisation of quantum hardware, today's quantum computers are qualified as Noisy Intermediate-Scale Quantum (NISQ) hardware. NISQ hardware is characterized by a small number of qubits (50 to a few hundred) and noisy operations. Moreover, current realisations of superconducting quantum chips do not have the ideal all-to-all connectivity between qubits but rather at most a nearest-neighbour connectivity. All these hardware restrictions add supplementary low-level requirements. They need to be addressed before submitting the quantum circuit to an actual chip. Satisfying these requirements is a tedious task for the programmer. Instead, the task of adapting the quantum circuit to a given hardware is left to the compiler. In this paper, we propose a Hardware-Aware mapping transition algorithm (HA) that takes the calibration data into account with the aim to improve the overall fidelity of the circuit. Evaluation results on IBM quantum hardware show that our HA approach can outperform the state of the art both in terms of the number of additional gates and circuit fidelity.

## 1 INTRODUCTION

In recent years, quantum computing has become a very active field of research. It promises to solve classically intractable computational problems such as integer factorisation [36], quantum chemistry [9], linear algebra [7, 18, 21, 22, 35, 43], or optimisation [15, 24, 25]. Along with algorithms, quantum hardware has attracted the attention of several companies such as IBM, Google, Intel, or Rigetti that have demonstrated quantum chips with 53, 72, 49, and 28 qubits respectively. IBM and Rigetti have also given access to a cloud quantum computing service on which anyone can submit quantum circuits to real quantum hardware. The aforementioned quantum hardware can already be qualified as NISQ hardware [33]. Still, none of them is fault-tolerant as quantum error correction codes (QECC) are in infancy. Nevertheless, it is believed that even a noisy quantum chip with limited qubit-to-qubit connectivity can be used to solve some classically intractable problems, one of the most promising candidates being quantum chemistry [9].

From the algorithm perspective, a new paradigm for quantum algorithms has emerged to take into account the limitations of NISQ hardware – variational algorithms. Examples of variational algorithms include the Variational Quantum Eigensolver (VQE) [32], the Variational Quantum Linear Solver (VQLS) [7, 22], or the Quantum Approximate Optimisation Algorithm (QAOA) [15]. However, there is a difference between the quantum program written by the programmer and what can be executed on the current quantum hardware. Quantum programs are written as if they were running on ideal quantum hardware without any noise or physical constraints. But real quantum chips are not ideal – for example for superconducting devices which are targeted in this paper, current two-qubit gates can at best only be applied between two neighbouring qubits. If we want to perform quantum computations, our quantum circuits must obey such connectivity constraints, which means that a modification of the quantum program is necessary to adapt it to the real quantum device. This problem of adapting a quantum program to given hardware connectivity is called the qubit mapping problem and is the focus of this paper.

The qubit mapping problem can be reformulated as two sub-problems. First, to find an initial mapping, i.e. a mapping between the "logical qubits" (as a qubit in a quantum circuit) to the "physical qubits" (as a qubit in a quantum chip). Secondly, to determine a mapping transition algorithm to identify the quantum gates to insert in a quantum circuit such that it complies with the targeted quantum hardware topology. Finding the optimal solution for the qubit mapping problem is likely to be an NP-complete problem as noted in [38].

Two types of methods have been used to solve the qubit mapping problem. The first method is to reformulate it as a mathematically equivalent problem that can then be solved using a specialised solver. Such mathematical formalism can be Integer Linear Programming (ILP) [4, 5, 13, 27], Satisfiability Modulo Theory (SMT) [30, 31], or even Constraint Programming (CP) [6, 40]. However, these mathematical approaches suffer from long runtime and are difficult to scale up. The second method is to use heuristics to modify the quantum circuit, starting from the first quantum gate and transforming the circuit sequentially by making each gate one after the other hardware-compliant.

Most of the previous works [2, 26, 34, 37, 42] using the second method only adapt for nearest-neighbour connectivity and cannot

be directly applicable to actual quantum architectures with nonuniform connections. Recently, publications[10, 19, 23, 29, 38, 44–46] that are not restricted to a specific architecture have been released. The algorithm presented in [46] uses a heuristic to find the best permutation at each step of the mapping procedure. Instead of representing a quantum circuit as a fixed sequence of layers, [20] introduces the Directed Acyclic Graph (DAG) that takes into account the dependency and commutativity of quantum gates. A major improvement has been shown by [29] which uses a "forward-backward-forward" mapping algorithm. Moreover, the "look-ahead" strategy has been introduced in the heuristic cost function for further optimisation in some existing works, notably [23, 29, 44–46]. For qubit movement, most of these methods only use `SWAP` gate. A notable exception is [23] that considered both `Bridge` and `SWAP` gate. Moreover, most of these works aim to minimise the number of inserted gates and do not consider the noise impact on different qubits.

In [3, 5, 16, 30, 39], calibration data is exploited and the applied approach is to insert additional gates between strongly linked qubits, i.e. qubits linked with a low two-qubit gate error rate. However, these works do not consider a holistic view of the problem such as exploring initial mapping or the heuristic function is not efficient enough to select the best candidate of the inserted gate.

In this work, we follow the second type of methods that consist in developing a heuristic to choose the best `SWAP` to insert based on calibration data. We propose a Hardware-Aware (HA) heuristic mapping transition algorithm to address the drawbacks mentioned above. Our main contributions can be listed as follows. First, we present a mapping transition algorithm that takes into account the hardware topology and the calibration data to improve the overall output state fidelity and reduce the total execution time. Second, to reduce the number of additional gates required to map the quantum circuit to the quantum chip, our algorithm can select between a `SWAP` or `Bridge` gate. Finally, we run our HA algorithm on real quantum hardware and compare with various mapping methods from the literature.

## 2 STATE OF THE ART

Here, we introduce state of the art on quantum hardware devices and their constraints, focusing mainly on IBM quantum devices. Then, we explain the qubit mapping problem. Finally, a small motivational example is shown to illustrate the gist of our algorithm. The notations used in this paper are summarised in Table 1 (we reference some notations from [29]).

### 2.1 Current state-of-the-art on quantum hardware

NISQ hardware is characterized in [33] as quantum hardware having from 50 to a few hundred noisy qubits on which one can only perform noisy operations. At the time of writing, several companies have already demonstrated quantum chips that can, according to the definition, be qualified as NISQ chips. For example, IBM announced the latest 53-qubit quantum chip and gave access to the community to execute quantum circuits. Other companies like Google (72 qubits) or Intel (49 qubits) announced quantum chips that could be qualified as NISQ but did not provide any information

**Table 1: Notations used in this paper**

| Notation | Definition |
|----------|-----------|
| $q$ | logical qubits for quantum circuit |
| $Q$ | physical qubits for quantum device |
| $g$ | quantum gate |
| $g.q_n$ | $n$-th logical qubit the quantum gate $g$ is applied on |
| $G$ | hardware coupling graph |
| $D$ | distance matrix |
| $S$ | swap matrix |
| $G_{\mathcal{E}}$ | hardware graph with swap error rates as weights |
| $\mathcal{E}$ | swap error matrix |
| $G_T$ | hardware graph with swap execution time as weights |
| $T$ | swap execution time matrix |
| $H$ | heuristic cost function |
| $\pi$ | mapping from $q$ to $Q$ |
| $F$ | first layer |
| $E$ | extended layer |
| $W$ | weight parameter |
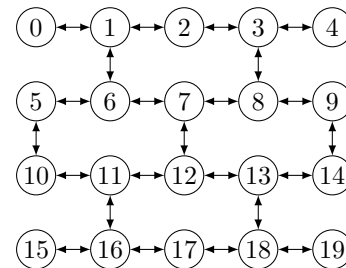


**Figure 1: ibmq_almaden topology. Qubits are represented as circles and indexed from 0 to 19. A connection between two qubits is represented by an edge between the two qubits.**

about their characteristics. One of the significant challenges of these quantum chips that limits them from solving real-world problems is their level of noise – even if these chips have enough qubits theoretically to show a quantum speedup, the fidelity of their quantum operations is still too low to obtain any advantage over the classical computer on real-world problems. In this paper, we mainly focus on IBM architectures. Other hardware such as Google's Sycamore [14] or Rigetti's Aspen-7 is not specially targeted. Still, the proposed algorithm and methods are general enough to be applicable to any quantum chip that use the quantum-gate model of computation and so should be applicable to these hardware.

Fig. 1 shows the topology, also called coupling graph, of IBM Quantum's ibmq_almaden, a 20-qubit system. Each vertex represents a qubit and the edge represents the coupling interconnect between two qubits. Table 2 shows the calibration data that are extracted from [1]. It includes `CNOT` error rates, single qubit error

**Table 2: ibmq_almaden characteristics**

| Qubit number | 20 |
|---|---|
| Single qubit error rate | $2.655e^{-4}$ to $9.569e^{-4}$ |
| CNOT error rate | $8.136e^{-3}$ to $3.403e^{-2}$ |
| `id` gate length | 35.56 ns |
| `u1` gate length | 0 ns |
| `u2` gate length | 35.56 ns |
| `u3` gate length | 71.11 ns |
| `CNOT` gate length | 248.88 ns to 860.44 ns |
| T1 | 34.66 $\mu$s to 139.46 $\mu$s |
| T2 | 12.16 $\mu$s to 200.25 $\mu$s |

Note that the exact hardware characteristics are not constant and change at each re-calibration of the chip.

rates, energy relaxation and decoherence characteristic times T1 and T2, and execution time (gate length). The calibration data show that the error of two-qubit gates is one order of magnitude higher than their one-qubit counterparts. This is also the case for gate execution times – two-qubit gates are approximately an order of magnitude slower than one-qubit gates. For simplicity and because of the relatively low error rates and execution times of one-qubit gates when compared to two-qubit gates, we focus on two-qubit gates in this paper.

Moreover, it is important to note that all the interconnects between qubits are not equal with respect to CNOT gate error rate or execution time. Taking ibmq_almaden as an example, the best CNOT gate has an error rate of 4.18 times lower than the worst CNOT and the maximum execution time is 3.46 times longer than the minimum one. Therefore, we cannot treat each qubit equally, and we need to consider the interconnect topology between qubits as well as their error rate. CNOT gates can be applied in either direction by conjugating with H gates. As we do not consider one-qubit gates in this study, we do not have to consider the connectivity direction.

## 2.2 Qubit mapping problem

Following the abstraction first introduced in classical computing decades ago, most of the quantum circuits are described in a generic manner that does not take into account all the physical hardware constraints. Many of the currently existing frameworks for quantum algorithm development encourage this way of development by giving access to a broad set of "primitive" gates. For example, the Qiskit library allows the developer to choose from more than 30 primitive gates, whereas the IBM quantum chips only provide four physical hardware gates (five if we take the identity gate into account). However, any gate can also be implemented with Open-Pulse [8] which is a low level hardware control for users to generate their gates to mitigate errors. Such an abstraction relieves the burden from the developer to adapt the code to a specific hardware and transfer it to the compiler, whose role is to transform an abstracted code into the most efficient hardware code possible. To do so, a compiler for quantum programs should perform several steps summarised in the following paragraphs.

The first step is to decompose the abstracted quantum gates into hardware gates. The hardware gates available strongly depend on the quantum hardware we are compiling for, but are generally comprised of a two-qubit "entangler" gate (controlled-X gate for IBM hardware, fSim gate for Google hardware) and several one-qubit gates (u1, u2, u3, and id gates for IBM hardware). At the end of this step, the quantum circuit has been modified to only contain quantum gates that are directly implemented in the quantum hardware.

But translating all abstract gates to hardware gates is generally not enough to make the quantum circuit executable on the specific hardware – the hardware topology is rarely respected at the end of this first step and the circuit requires a second step with further modifications. Such modification of the quantum circuit to make it compliant with the hardware topology is often done by inserting SWAP gates before non-executable two-qubit gates. Note that on current hardware, only two-qubit gates are restricted by the hardware topology.

Finally, once the quantum circuit is executable on the specified quantum hardware, a final third step is performed to optimise the quantum circuit. Depending on the figure of interest, the optimisation can aim at reducing the execution time, gate count, increasing the final state fidelity or even reducing the number of qubits needed.

The qubit mapping problem is defined as the second compilation step that modifies the quantum circuit to contain only two-qubit gates that fit into the hardware topology. But in practice qubit mapping algorithms also try to consider the third step that consists in optimising the generated quantum circuit according to a chosen figure of merit.

## 2.3 Motivational example

Fig. 2a shows a small quantum circuit, which is composed of three CNOT gates and one X gate. It is mapped to a 5-qubit IBM quantum device called ibmq_valencia, shown in Fig. 2b. For simplicity, the initial mapping is allocated linearly as $\{q_0 \rightarrow Q_0, q_1 \rightarrow Q_1, q_2 \rightarrow Q_2, q_3 \rightarrow Q_3, q_4 \rightarrow Q_4\}$. Gates $g_1$ and $g_2$ comply with the hardware topology (i.e. coupling constraints) and can be executed directly. However, $g_3$ is applied to two non-connected qubits. Therefore, a movement (i.e. a SWAP gate, shown in Fig. 3) of logical qubits is needed before being able to execute $g_3$ on the hardware connection between $q_2$ and $q_3$. Referring to the coupling graph in Fig. 2b, three SWAP gates are possible: $\{q_1, q_2\}$, $\{q_1, q_3\}$ and $\{q_3, q_4\}$. Among these possible SWAPs, two of them change the current mapping between logical and physical qubits in such a way that the CNOT gate between $q_2$ and $q_3$ becomes executable – swapping of $\{q_1, q_2\}$ and $\{q_1, q_3\}$. Translating the logical qubits to their physical counterparts, the SWAPs $\{Q_1, Q_2\}$ and $\{Q_1, Q_3\}$ are our candidates. At this step, most of the state-of-the-art algorithms consider the two possible SWAPs to be equal and will randomly select one. However, if the calibration data is considered, the SWAP between $\{Q_1, Q_2\}$ is less noisy than the other (error rate of the two interconnects is shown in Fig. 2b). A SWAP operation consists of three CNOTs and we want to insert a SWAP gate with the least noise. Thus, the SWAP gate between $\{q_1, q_2\}$ is inserted and the final mapping is $\{q_0 \rightarrow Q_0, q_1 \rightarrow Q_2, q_2 \rightarrow Q_1, q_3 \rightarrow Q_3, q_4 \rightarrow Q_4\}$. The updated circuit is shown in Fig. 2c.
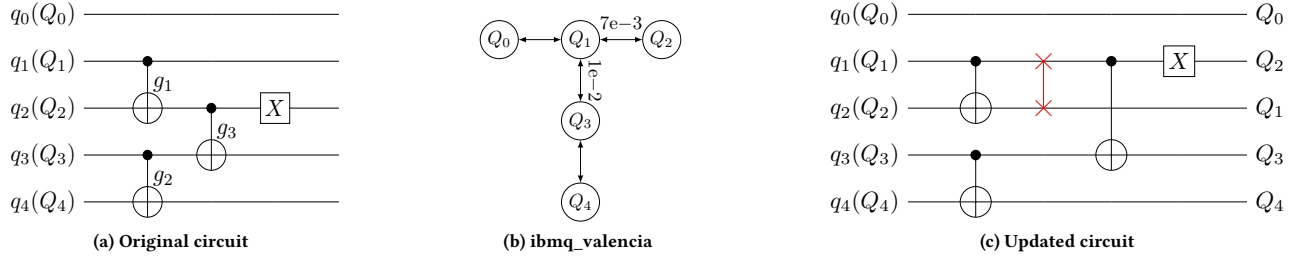
(a) Original circuit      (b) ibmq_valencia      (c) Updated circuit

**Figure 2: A motivational example for the qubit mapping problem.**

## 3 PROPOSED SOLUTION

We are inspired by the SABRE algorithm presented in [29], which is a SWAP-based heuristic algorithm to reduce the number of additional CNOT gates. We propose a Hardware-Aware SWAP and Bridge based heuristic search algorithm. Compared to SABRE algorithm, which aims at reducing the number of additional gates, we improve the circuit fidelity as well as reduce the number of additional gates by introducing a new distance matrix that takes into account both of the hardware connectivity and the calibration data. Moreover, SABRE only uses SWAP gate when a qubit movement is needed, whereas our algorithm decides between a SWAP and Bridge for qubit movement to further reduce the number of additional gates. Finally, we also develop an initial mapping algorithm called Hardware-aware Simulated Annealing (HSA) in order to evaluate the mapping transition algorithm of different flavours.

The compiler takes as input a quantum program written in the OpenQASM language [11] and the calibration data of a specific IBM quantum device. During the compilation process, it considers the hardware constraints such as hardware topology and gate availability. Then, the qubit mapping algorithm is applied. It contains two principal parts – initial mapping and mapping transition algorithm. In the mapping transition step, some optimisations are done to generate a circuit with a better performance in terms of final state fidelity. The source code is publicly available at https://github.com/peachnuts/HA.

We start by explaining our HA algorithm in subsection 3.1. In subsection 3.2, we describe the hardware-aware simulated annealing (HSA) method for initial mapping. Finally, subsection 3.3 presents the metrics used to evaluate our algorithm.

### 3.1 Hardware-aware SWAP and Bridge based heuristic search

The first step of the algorithm is to process the input quantum circuit in order to reformulate it in a more convenient data format. Starting from the input quantum circuit, we can obtain a Directed Acyclic Graph (DAG) circuit which represents the operation dependencies in the quantum circuit without considering the hardware constraints. The DAG is constructed such that quantum gates are represented by the graph nodes and the directed edge $(i, j)$ between nodes $i$ and $j$ represents a dependency from gate $i$ to $j$, i.e. gate $i$ should be executed before $j$.

Once the DAG is constructed, graph nodes (i.e. quantum gates) can be ordered according to the gate dependencies âĂŞ for example
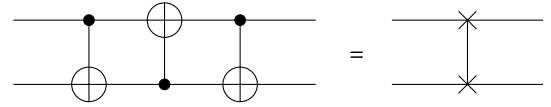


**Figure 3: SWAP gate**

if gate $j$ depends on gate $i$, then gate $i$ will be ordered before gate $j$. One possible ordering that fulfil this property of dependency is the well known topological ordering. Note that depending on the quantum circuit, this ordering might not be unique.

Quantum gates can then be divided into three groups: the executed gates, the executable gates, and to be executed future gates. Executed gates are quantum gates that have already been mapped by the algorithm. Executable gates constitute the first layer, denoted $F$. A gate is considered executable when all the gates it depended on are in the executed gates group. Finally, to be executed future gates are the rest of the gates (not yet executed nor executable). These gates are included in the extended layer, $E$. An illustration of layers $E$ and $F$ is shown in Fig. 6.

*3.1.1 Heuristic cost function.* A heuristic cost function $H$ is introduced to estimate the cost of each possible (i.e. executable) swap pairs at a given step of the iterative algorithm. Its objective is to quantify the quality of the possible swap pairs according to the distance considered and to select the best swap pair.

When inserting a SWAP gate, the circuit is divided into two layers: the first layer $F$ and the extended layer $E$. Note that inserting a SWAP gate will not only influence the gates in the first layer $F$ but also the gates in the extended layer $E$. The approach of considering the swap pair's impact on the extended layer is referred as the look-ahead ability. It can contribute to a better selection and depends on the size of the extended layer.

We devise several metrics that can be used to estimate the cost of a swap pair in HA. We consider three different distance matrices – swap matrix $S$, swap error matrix $\mathcal{E}$ and swap execution time matrix $T$. Because $S$, $\mathcal{E}$, and $T$ contain entries with incompatible units and different scales, we update $T$ to make it dimensionless and each matrix is normalised. Moreover, we introduce weights ($\alpha_1$, $\alpha_2$, and $\alpha_3$ for $S$, $\mathcal{E}$, and $T$, respectively) to allow to choose the importance of each parameter in terms of number of SWAPs, gate error and execution time.

Matrix $S$ is constructed such that the entry $(i, j)$ stores the distance on the real hardware between qubit $i$ to a neighbour of qubit

$j$, which is also equal to the minimum number of SWAP gates needed to move qubit $i$ to qubit $j$. The matrix is efficiently constructed by using the Floyd-Warshall algorithm [17].

Matrix $\mathcal{E}$ stores in its entry $(i, j)$ the minimum error rate attainable to move the qubit $i$ to a neighbour of qubit $j$. The error rate of each possible SWAP is computed based on the calibration data of CNOT gates. The decomposition of a SWAP gate in terms of CNOT gates is shown in Fig. 3.

The success rate of a CNOT between the physical qubits $Q_i$ and $Q_j$, denoted by $S(Q_i, Q_j)$, is computed from the error rates given in the calibration data. Equation (1) computes the error rate of a SWAP gate between two connected physical qubits $Q_i$ and $Q_j$ while taking into account that the swap operation is symmetric. The final $\mathcal{E}$ matrix is constructed by using the Floyd-Warshall algorithm on the graph $G_{\mathcal{E}}$ with the computed errors as edge weights.

$$G_{\mathcal{E}}(Q_i, Q_j) = 1 - S(Q_i, Q_j) \times S(Q_j, Q_i)$$
$$\times \max(S(Q_i, Q_j), S(Q_j, Q_i)) \quad (1)$$

Matrix $T$ is computed, similarly as $S$ and $\mathcal{E}$, with the Floyd-Warshall algorithm applied on graph $G_T$ but by using the SWAP execution time. This execution time is computed with (2) where $t(Q_i, Q_j)$ is the execution time of the CNOT gate with $Q_i$ as control and $Q_j$ as target, extracted from the calibration data.

$$G_T(Q_i, Q_j) = t(Q_i, Q_j) + t(Q_j, Q_i)$$
$$+ \min(t(Q_i, Q_j), t(Q_j, Q_i)) \quad (2)$$

The summation of the three matrices forms a new matrix called distance matrix $D$ (shown in (3)). The distance matrix represents the "distance" between each pair of qubits in the quantum chip. Here, the "distance" means the combination of swap distance, overall error rate and execution time of the shortest path.

$$D = \alpha_1 \times S + \alpha_2 \times \mathcal{E} + \alpha_3 \times T \quad (3)$$

Inserting a SWAP gate will have an impact on the current mapping $\pi_c$, changing it to $\pi_{\text{temp}}$. We compute the cost of this SWAP on the first layer $F$ with the cost function $H_{basic}$ shown in (4). A small score means the SWAP has a little impact on the first layer gates with respect to the overall distance considered. The swap pair with the minimum score is selected as the best candidate.

$$H_{basic} = \sum_{g \in F} D[\pi_{\text{temp}}(g.q_1)][\pi_{\text{temp}}(g.q_2)] \quad (4)$$

We also consider the impact of the swap pair on the extended layer $E$. The impact of a SWAP on the first layer is prioritised over its impact on the extended layer. As a result, a weight parameter $W$ is added to the extended layer cost to scale its impact. Moreover, the impacts on the first layer and extended layer are normalised by dividing them with their respective number of gates. The complete heuristic function including the extended layer $E$ with look-ahead ability is shown in (5). Even though (4) and (5) are similar to equations in [29], it is important to note that the distance matrix $D$ is different.
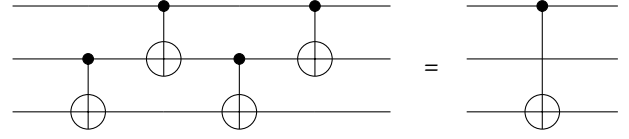


Figure 4: Bridge gate

$$H = \frac{1}{|F|} \sum_{g \in F} D[\pi_{\text{temp}}(g.q_1)][\pi_{\text{temp}}(g.q_2)]$$
$$+ W \times \frac{1}{|E|} \sum_{g \in E} D[\pi_{\text{temp}}(g.q_1)][\pi_{\text{temp}}(g.q_2)] \quad (5)$$

*3.1.2 SWAP gate and Bridge gate.* Another important metric of the HA algorithm is the heuristic cost function that estimates the usefulness of a SWAP. In some situations, even the best SWAP may have a negative impact on the overall circuit. In that case, inspired by [23], our heuristic function decides to insert a Bridge gate instead of a SWAP gate if the topology allows it. The decomposition of the Bridge gate with four CNOTs is shown in Fig. 4. The Bridge gate allows executing a CNOT between two qubits that share a common neighbour. Both SWAP and Bridge gate need three supplementary CNOTs. Note that the Bridge gate can only be used to replace a CNOT if the distance between the control and target qubits (i.e. the minimum number of links between the two qubits) is exactly two.

Fig. 5a shows an example of quantum circuit that is mapped to ibmq_valencia with the topology described in Fig. 2b. The quantum gates $g_1$ and $g_2$ comply with the topology of the chip, but $g_3$ does not. By evaluating the heuristic cost function $H$, the SWAP between $q_0$ and $q_1$ is selected. But as shown in Fig. 5b, the chosen SWAP has a negative impact on the extended layer – gate $g_5$ is no longer executable and another SWAP gate is required to execute it.

Such situations can be solved by using a Bridge gate instead of a SWAP gate as shown in Fig. 5c. Since the distance between the control qubit $q_0$ and the target qubit $q_3$ of gate $g_3$ is two, we can insert a Bridge gate instead. Using a Bridge gate allows to execute the CNOT gate $g_5$ without changing the current mapping. Moreover, by using a Bridge gate, we only add three CNOTs to map the entire circuit, instead of six (two times more) if only SWAP gates were used.

Once the cost $H$ of each swap pair is computed, the heuristic will try to choose the best option between inserting a SWAP or Bridge gate. To do so, it considers two mappings: $\pi_c$, the mapping used before selecting the best swap pair and also the mapping obtained after inserting a Bridge gate, and $\pi_{\text{temp}}$, the new mapping that would be obtained after inserting the best SWAP gate. The overall effect of the SWAP gate on the extended layer $E$ is computed according to (6). If the effect of the best SWAP gate is negative, this means that the considered swap pair has an overall negative impact on the extended layer $E$. In this case, we consider that it is better to keep the current mapping so, if the hardware topology permits it, a Bridge gate is inserted instead of a SWAP gate.
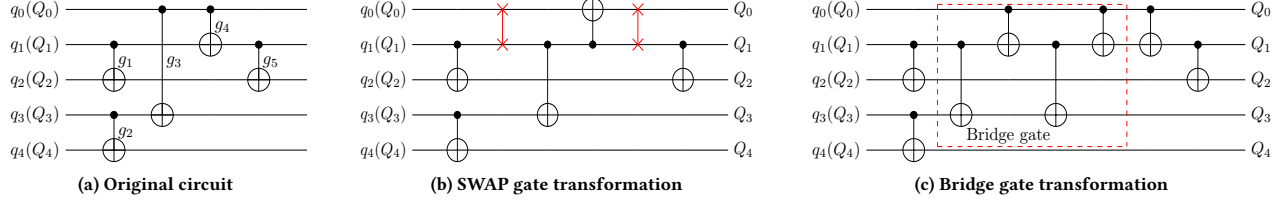
**(a) Original circuit**          **(b) SWAP gate transformation**          **(c) Bridge gate transformation**

**Figure 5: An example of a quantum circuit showing the difference between SWAP and Bridge transformation.**

$$\text{Effect} = \sum_{g \in E} D[\pi_c(g.q_1)][\pi_c(g.q_2)]$$
$$- D[\pi_{\text{temp}}(g.q_1)][\pi_{\text{temp}}(g.q_2)] \quad (6)$$

*3.1.3 HA Algorithm.* The mapping transition algorithm will go through each quantum gate sequentially and mark the directly executable gates as executed. If no more gates can be marked as *executed*, this means that either the quantum circuit is fully mapped or all the gates in the first layer do not comply with the hardware topology. In the first case, the mapping algorithm can be stopped and the mapped quantum circuit returned. In the second case, the algorithm calls a heuristic function to choose the best SWAP or Bridge gate to insert in order to make some of the gates in the first layer executable. The algorithm then iterates, until the quantum circuit is fully mapped.

Algorithm 1 shows the pseudo-code of the HA heuristic method. Note that the most recent calibration data should be retrieved (i.e. through the IBM Quantum Experience) before each usage of the HA algorithm to ensure that the algorithm has access to the most accurate and up-to-date information possible.

The heuristic method to insert a SWAP or Bridge when no gate in the first layer $F$ is executable can be described as follows. First, a list of all the candidate SWAP gates, swap_candidate_list, is constructed based on the quantum gates in the first layer $F$ and the hardware coupling graph $G$. Then, for each SWAP candidate a temporary mapping $\pi_{temp}$ is computed with the Map_Update function. The final cost of the candidate SWAP is computed following (5). The SWAP with the minimum score is selected and called $swap_{min}$.

The last step is to choose between a SWAP gate or a Bridge gate. A SWAP gate can always be used, whereas a Bridge gate can only be inserted if a gate in the first layer $F$ becomes executable from the mapping obtained after applying the $swap_{min}$ gate. If a Bridge gate is not insertable, then the algorithm has no choice but to insert a SWAP gate. Else, the algorithm decides the gate (SWAP or Bridge) to insert based on the effect of the SWAP gate on the extended layer computed with (6). If adding a SWAP gate has a negative impact on the extended layer, then a Bridge gate (which does not change the current mapping) is inserted. Otherwise, if adding a SWAP gate has a positive effect on the extended layer, then the algorithm inserts a SWAP gate.

*3.1.4 Runtime analysis.* The HA algorithm outperforms SABRE algorithm thanks to several modifications while not changing its asymptotic complexity. The mapping procedure is separated into

---

**Algorithm 1:** Heuristic algorithm for selecting additional gate candidate

**input** : Circuit $DAG$, Coupling graph $G$, Current mapping $\pi_c$, Distance matrix $D$, Swap matrix $S$, First layer $F$, Extended layer $E$, Weight parameter $W$

**output**: New mapping $\pi_n$, Inserted gate $g_{\text{add}}$

1 **begin**
2      Set score to empty list;
3      Set effect to empty list;
4      swap_candidate_list ← FindSwapPairs($F, G$);
5      **for** swap ∈ swap_candidate_list **do**
6          $\pi_{\text{temp}}$ ←Map_Update (swap);
7          $H_{\text{basic}} \leftarrow 0$;
8          **for** gate ∈ $F$ **do**
9              $H_{\text{basic}} \leftarrow H_{\text{basic}} + D(\text{gate}, \pi_{\text{temp}})$ ;
10          **end**
11          $H_{\text{extended}} \leftarrow 0$;
12          **for** gate ∈ $E$ **do**
13              $H_{\text{extended}} \leftarrow H_{\text{extended}} + D(\text{gate}, \pi_{\text{temp}})$ ;
14              effect_cost ← effect_cost + $D(\text{gate}, \pi_c) - D(\text{gate}, \pi_{\text{temp}})$;
15          **end**
16          $H \leftarrow \frac{1}{|F|}H_{\text{basic}} + \frac{W}{|E|}H_{\text{extended}}$ ;
17          score.append($H$);
18          effect.append(effect_cost);
19      **end**
20      *Find the swap with minimum score:* swap$_{\text{min}}$;
21      *Find the gate in $F$ that become executable by applying* swap$_{\text{min}}$: $g_s$;
22      **if** effect $\big[$swap$_{\text{min}}\big] < 0$ **and** $S(g_s, \pi_c) = 2$ **then**
23          $\pi_n \leftarrow \pi_c$;
24          $g_{\text{add}} \leftarrow g_B$;
25      **else**
26          $\pi_n \leftarrow$ Map_Update (swap$_{\text{min}}$);
27          $g_{\text{add}} \leftarrow$ swap$_{\text{min}}$;
28      **end**
29      **return** $\pi_n, g_{\text{add}}$;
30 **end**

**(a)** Beginning of the HA mapping algorithm. $g_1$ and $g_2$ do not overlap and are the first gates in the circuit so they are in $F$. The other gates are pushed in $E$.

**(b)** $g_1$ and $g_2$ are compliant with the hardware topology. They are executed and removed from $F$. The gate $g_3$ is pushed into $F$ but is not compliant and a SWAP/Bridge should be inserted. $g_4$ overlaps with $g_3$ and cannot be inserted in $F$.

**(c)** After Bridge insertion, $g_3$ is executed and removed from $F$. $g_4$ no longer overlap with a gate in $F$ and is added to the first layer. $g_5$ overlaps with $g_4$ and so should stay in $E$.
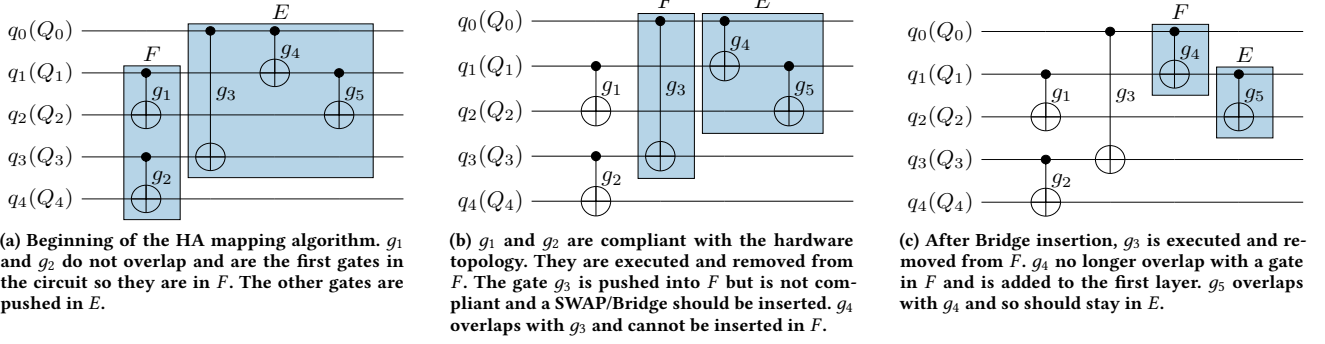
**Figure 6: Evolution of the layers $F$ and $E$ on a simple circuit with a detailed explanation at each step.**

two steps: an initialisation step that is independent of the mapped quantum circuit and a mapping step.

The initialisation step computes the distance matrix that is used afterwards in the mapping step. In our algorithm, the distance matrix is computed according to (4). Each of $S$, $\mathcal{E}$ and $T$ constituting the distance matrix $D$ requires to use the Floyd-Warshall algorithm once on the hardware graph. This means that we need to perform three calls to an algorithm of $O(n^3)$ complexity, $n$ being the number of qubits of the targeted quantum chip. Moreover, the weights used by the Floyd-Warshall algorithm for the matrices $\mathcal{E}$ and $T$ should be retrieved online with Qiskit API. This retrieval is an operation that theoretically takes $O(n^2)$ time in the worst case as we need to retrieve CNOT error rates and execution time for each link. Note that the current quantum chips only have $O(n)$ links and so the asymptotic complexity of this step is $O(n)$. Overall, the initialisation step is dominated by the cost of applying the Floyd-Warshall algorithm, that takes $O(n^3)$ time.

After the initialisation step, the actual mapping procedure is applied. Let $n$ be the number of qubits, $g$ the number of CNOT gates in the mapped quantum circuit and $d$ the diameter of the chip, i.e. the minimum SWAP distance between the two farthest qubits on the quantum chip. In the worst case, all the CNOT gates should be mapped because none of them comply with the hardware topology. Moreover, all the CNOT gates might need up to $d$ SWAPs in order to become executable. Finally, for each SWAP insertion we need to execute the heuristic cost function. This function will need to explore at most $n^2$ links (in the case of an all-to-all connected chip, this number improves to $O(n)$ on practical quantum chips with a nearest-neighbour connectivity), where exploring one link might take a time of $O(g)$ if all the CNOT gates are included in either $F$ or $E$. In summary, the mapping step takes $O(g^2 d n^2)$ time in the worst case, which can be improved to $O(g n^{2.5})$ under reasonable assumptions (nearest-neighbour chip connectivity, i.e. $d \in O(\sqrt{n})$, and an extended layer $E$ with at most $O(n)$ CNOT gates).

It is important to note that the initialisation step only needs to be repeated when the calibration data change but that requires to recover data from the Internet which can be a slow operation (in the order of several seconds).

## 3.2 Initial mapping

Heuristic-based mapping transition algorithms rely crucially on a good initial mapping to achieve the best results. A well-known algorithm when trying to approximate the global minimum of a scalar function with a discrete search space is simulated annealing. Simulated annealing is a meta-heuristic designed to explore the search space by randomly selecting neighbours of the current state, evaluating them with the provided cost function and evolving in such a way that the algorithm will not be trapped into local minimums. The simulated annealing algorithm is depicted in Algorithm 2.

A modified version of simulated annealing has already been applied in [44] where a repetition parameter $R$ is used to explore several neighbours at each temperature step. The authors consider a simple `get_neighbour` function that modifies randomly the current mapping $\pi$ to a neighbouring mapping $\pi_{\text{neighbour}}$. However, `get_neighbour` function is limited as it is not aware of the underlying hardware. This means that from the set of mappings generated by this function and evaluated by the simulated annealing procedure, several mappings can be excluded even before evaluating the mapping cost.

We aim to improve the initial mapping generated with the simulated annealing procedure by designing a Hardware-aware Simulated Annealing (HSA) algorithm using a hardware-aware `get_neighbour` method to explore the neighbouring mappings. To explore different mappings, we separate the `get_neighbour` procedure in three algorithms governed by a top execution policy. This top layer policy decides which one of the three algorithms the `get_neighbour` method should execute to obtain a new mapping. The policy we used randomly chooses which algorithm to use from the value of a random number.

The first algorithm, called `shuffle`, does not change the physical qubits involved in the current mapping but changes how they are mapped to logical qubits. The most straightforward algorithm that can be used for this task is a random shuffle – we list the physical qubits involved in the mapping, randomly shuffle them, and obtain a new arbitrary mapping with the same physical qubits.

The second algorithm, expand, does not change the mapping between physical qubits and logical ones but replaces one of the physical qubits involved in the mapping by another physical qubit

**Algorithm 2:** Simulated annealing

**input** : Initial mapping $\pi_0$, Cost function C, Neighbour computation function `get_neighbour`, Initial temperature $T_{\text{init}}$, Final temperature $T_f$, Temperature evolution constant $\Delta$
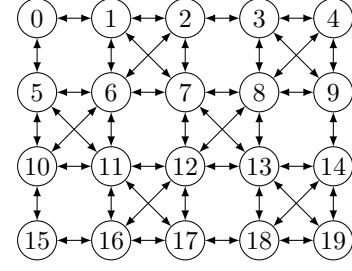
**output**: Best initial mapping found $\pi_{\text{opt}}$

```
1  begin
2  |   π ← π₀;
3  |   π_opt ← π₀;
4  |   T ← T_init;
5  |   cost ← C(π);
6  |   cost_opt ← cost;
7  |   while T ⩾ T_f do
8  |   |   π_neighbour ← get_neighbour(π);
9  |   |   cost_neighbour ← C(π_neighbour);
10 |   |   if cost_neighbour < cost_opt then
11 |   |   |   cost_opt ← cost_neighbour;
12 |   |   |   π_opt ← π_neighbour;
13 |   |   end
14 |   |   if cost_neighbour < cost then
15 |   |   |   cost ← cost_neighbour;
16 |   |   |   π ← π_neighbour;
17 |   |   else
18 |   |   |   if rand() < exp( (cost - cost_neighbour)/T ) then
19 |   |   |   |   cost ← cost_neighbour;
20 |   |   |   |   π ← π_neighbour;
21 |   |   |   end
22 |   |   end
23 |   |   T ← T × Δ;
24 |   end
25 |   return π_opt;
26 end
```



**Figure 7: ibmq_tokyo topology.**

initial mapping from a randomly chosen qubit, without considering the previously explored mappings. The algorithm starts with a random qubit and expands the mapping by iteratively weighting all the qubits and adding the best qubit to the new mapping.

### 3.3 Metrics

To evaluate our solution and compare it to other algorithms, we use some metrics that are described in the following paragraphs.

The first metric is the success rate of the mapped quantum circuit on a given hardware. We define the success rate of a quantum circuit as the fidelity of the quantum state obtained at the end of the execution of this quantum circuit on the hardware. We estimate this success rate by executing the quantum circuit a large number of times (8192), counting the number of executions that gave the expected answer and dividing this number by the total number of executions. The expected answer is obtained by executing the quantum circuit on a simulator.

The second metric chosen is the additional number of CNOT gates. This metric is tightly linked with the total number of SWAP gates inserted.

The third metric is the total execution time of the circuit. As the execution time of each CNOT gate can be extracted from [1], we can estimate the overall execution time for a given circuit. This metric is important for several reasons. First, it shows the ability of the mapping algorithm to schedule gates in parallel when possible and how good is the algorithm at doing this. Secondly, it allows us to have an idea of the importance of decoherence noise in the computed fidelity. For each qubit, the execution time is computed by adding the total execution time of gate operations acting on it. The longest qubit execution time is selected to represent the total execution time of the quantum circuit.

## 4 EVALUATION AND COMPARISON OF THE PROPOSED HA ALGORITHM

### 4.1 Methodology

All the benchmarks used are collected from the previous works [28, 29, 46]. They include several functions taken from RevLib [41] as well as quantum algorithms from a variety of domains including optimization, simulation, quantum arithmetic, etc. They are well known in the community and given as quantum circuits written in the OpenQASM language [11].

that is not part of the mapping. Instead of a hardware-unaware expand, we use an expand algorithm that tries to avoid separating the physical qubits in the current mapping into two disconnected groups. Moreover, the algorithm encourages re-arrangement of qubits based on the figure of merit chosen (i.e. final state fidelity, circuit depth, execution time). In this algorithm, we consider that strongly connected qubits have high fidelity. The hardware-aware implementation aims to identify the qubits with the least and most connections. Moreover, based on our tests with qubit measurement operations, we find there is a huge source of errors. To account for theses errors, we add weights to qubit to determine the best and worst qubits in terms of their measured fidelity.

The third algorithm used in the `get_neighbour` algorithm is called `reset`. Its purpose is to give the possibility to the simulated annealing algorithm to escape local-minimums. This algorithm is needed because the first two algorithms `shuffle` and `expand` will likely explore only the close neighbourhood of the current mapping and may not be able to escape a local minimum. To avoid being stuck, the `reset` algorithm tries to find a potentially good new

(a) Number of additional gates

(b) Fidelity

Figure 8: Comparison of number of additional gates and fidelity on ibmq_valencia. HA has been used with $\alpha_1 = 0.5$, $\alpha_2 = 0.5$ and $\alpha_3 = 0$.



(a) Number of additional gates

(b) Fidelity

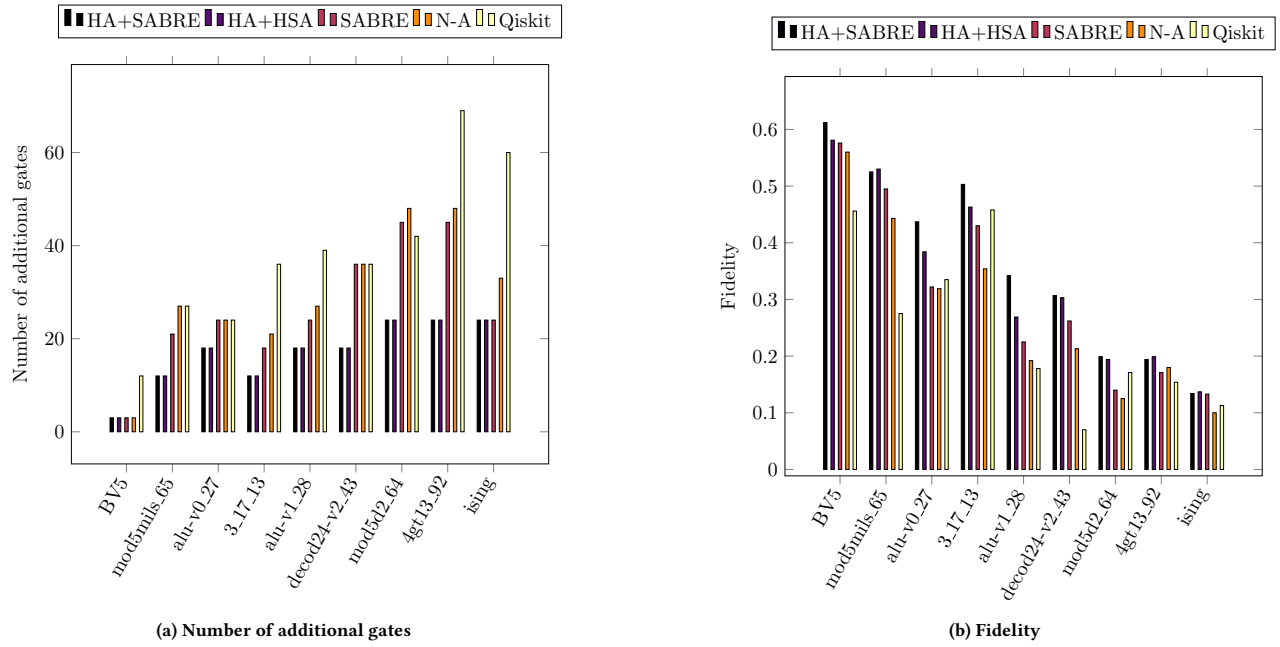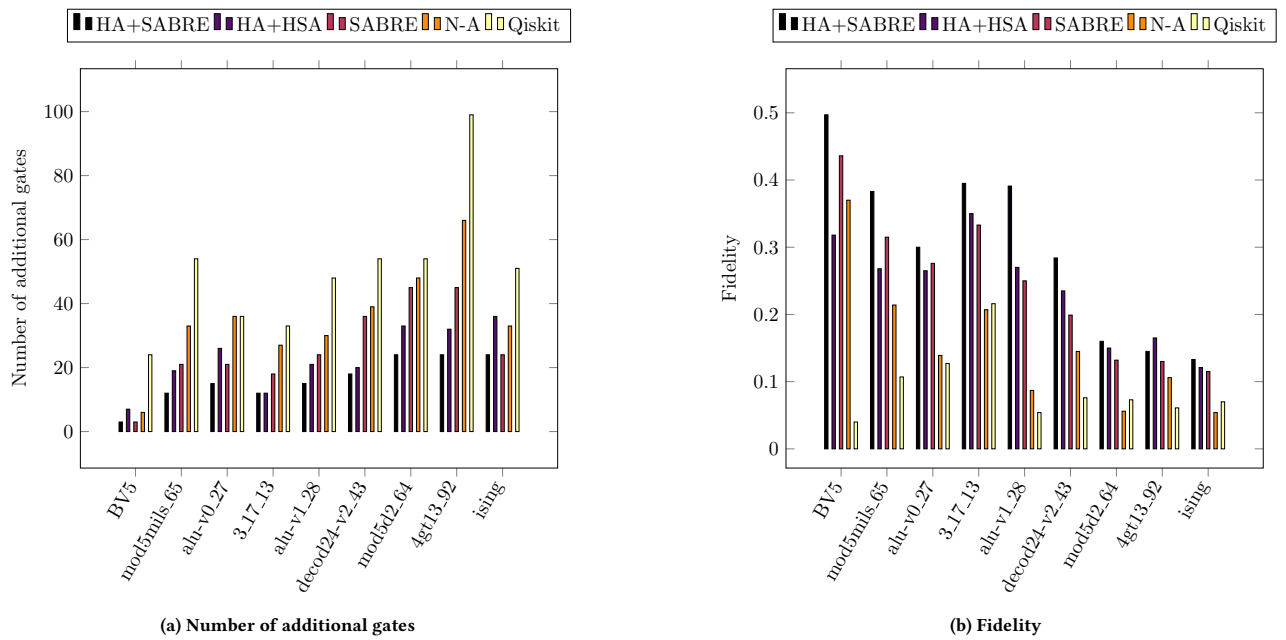Figure 9: Comparison of number of additional gates and fidelity on ibmq_almaden. HA has been used with $\alpha_1 = 0.5$, $\alpha_2 = 0.5$ and $\alpha_3 = 0$.

We chose two quantum chips, ibmq_almaden and ibmq_valencia, available from IBM Quantum experience website and one quantum chip ibmq_tokyo, which is not accessible currently but widely used by state-of-art algorithms. ibmq_almaden is a 20-qubit quantum chip. Its topology and characteristics are summarised in Fig. 1 and Table 2. ibmq_valencia is a 5-qubit chip depicted in Fig. 2b. ibmq_tokyo is a 20-qubit virtual chip depicted in Fig. 7. We execute benchmarks on ibmq_valencia and ibmq_almaden to check cases when the mapped quantum circuit needs all the available qubits or only a small number of them. Moreover, we use ibmq_almaden and ibmq_tokyo to compare our algorithm with state-of-art algorithms in terms of number of additional gates. Note that we do not have to execute the mapped quantum circuit on real quantum hardware to count the number of additional gates.

Our algorithm is implemented in Python and the Qiskit version is 0.19.1. To empirically evaluate our algorithm, we use a personal computer with 1 Intel i5-5300U CPU and 8 GB memory. The Operating System is Ubuntu 18.04.

Several published qubit mapping algorithms are available as discussed in section 1. SABRE [29] seems to be the best algorithm at the time of writing when comparing the number of inserted gates to make the quantum circuit hardware-compliant. It provides a good initial mapping method and a mapping transition algorithm. Another algorithm DL [45] (Dynamic look-ahead) based on SABRE shows an improvement in terms of number of additional gates. Moreover, the mapping method presented in [30] uses the hardware calibration data to try to find a good mapping. We compare to all these algorithms. The source code of SABRE has been provided by the authors of the algorithm, and the mapping method presented in [30], called Noise-Adaptive (N-A) Compiler, has been integrated into Qiskit as a transpiler pass. Finally, we also include the default transpiler included in Qiskit as the baseline. We execute our HA mapping transition algorithm with two different initial mapping algorithm – SABRE initial mapping algorithm and our Hardware-aware Simulated Annealing (HSA) algorithm.

Summarising, to test on real hardware, five different algorithms are included in the benchmarks: 1) our HA mapping algorithm with SABRE initial mapping, 2) our HA mapping algorithm with HSA initial mapping, 3) SABRE mapping algorithm with SABRE initial mapping, 4) N-A Compiler and 5) Qiskit transpiler. For a fair comparison, we set the `optimisation_level` parameter of the Qiskit transpiler to zero and make sure that the circuits obtained from the five methods are all executed with the same calibration data. The `optimisation_level` is set to zero to invoke only the mapping transformation and not the optimisation transformation. Moreover, when using the N-A Compiler, the routing method is set to "lookahead" to make sure that it has the look-ahead ability. To compare the number of additional gates without accessing to real hardware, three algorithms are included: 1) SABRE, 2) DL, 3) HA.

To evaluate our algorithm with the different initial mapping methods, we allow each of them to call the mapping algorithm at most 100 times. The number of calls to the mapping algorithm is a natural parameter of the simulated annealing-based method, but the SABRE initial mapping method only needs two calls. To let the SABRE algorithm take advantage of a larger number of calls, we repeat the algorithm on several random initial mappings until no

more calls are allowed and choose the best mapping found. The whole process is repeated 10 times to obtain 10 initial mappings.

We divide benchmarks by size according to their number of gates. We only execute small size benchmarks on real quantum hardware, because the other benchmarks with a large number of gate operations introduce too much noise to obtain any meaningful results. Moreover, the initial mapping generation process described above is applied on small and medium sized benchmarks. Large benchmarks suffer from long run time, so we generate 10 initial random mappings and use them with different algorithms. When using ibmq_tokyo virtual chip, we select the best results out of five attempts which is a similar approach applied in SABRE and DL.

When testing the HSA algorithm we used a random policy to choose which one of the three subroutines to execute. The `shuffle` procedure is executed with a probability of 0.9, the `expand` algorithm is chosen with a probability 0.08 and the `reset` procedure is executed when the two previous algorithms are not used (i.e. 0.02 probability).

First, we compare the number of additional gates and fidelity. The weight parameter $\alpha_1$ of swap matrix $S$ is set to 0.5, the weight parameter $\alpha_2$ of CNOT error matrix $\mathcal{E}$ is set to 0.5 and the weight parameter $\alpha_3$ of CNOT execution time $T$ is set to 0. Second, we compare the number of additional gates and total execution time. Weight parameter $\alpha_1$ of swap matrix $S$ is set to 0.5, weight parameter $\alpha_2$ of CNOT error matrix $\mathcal{E}$ is set to 0 and weight parameter $\alpha_3$ of CNOT execution time $T$ is set to 0.5. Third, we compare the number of additional gates for circuits that are not executable on the real quantum device. Weight parameter $\alpha_1$ of swap matrix $S$ is set to 1, and the other two parameters are set to 0. For these three comparisons, the weight parameter $W$ in the cost function is set to 0.5 and size of extended layer is set to 20.

## 4.2 Experimental results

We compare both the average number of additional gates (see Fig. 8a and Fig. 9a) and average output state fidelity (see Fig. 8b and Fig. 9b) among the 10 initial mappings for the five methods. The complete experimental results are listed in Table 5 and Table 6.

The Qiskit default qubit mapping algorithm is nearly always the worst one in terms of additional gates, which translates in most of the cases to the worst output state fidelity. Although N-A Compiler takes into account the calibration data and has the look-ahead ability, results show that it does not outperform the SABRE mapping algorithm with SABRE initial mapping (labelled as SABRE in the plots). Our HA mapping algorithm with SABRE initial mapping (labelled as HA+SABRE in the plots) seems to be the best combination as in average it achieves the best output state fidelity. Moreover, our HA algorithm with SABRE initial mapping gives the minimum number of additional gates. HA mapping algorithm with HSA initial mapping (labelled as HA+HSA in the plots) is also good, but its results are less consistent than HA+SABRE due to its random nature. Although, in many test cases, it outperforms SABRE.

We also tried to map and execute the qft_10 circuit. We found that its output fidelity is less than 0.01 for all the methods tested in the benchmark. Because the base fidelity is too low to perform a meaningful comparison, we only compare the number of additional

gates as summarised in Table 3 and Table 4 for quantum circuits with a medium-to-large number of gates.

Fig. 10 shows the result of comparing the execution times, number of additional gates and fidelities of our HA algorithm with SABRE algorithm on ibmq_valencia. The execution time is reduced by 19% on average. Even though the weight parameter $\alpha_2$ of CNOT error matrix $\mathcal{E}$ is set to 0, the fidelity is improved by 8%. The number of additional gates is reduced by 38%.

Table 3 lists the result of the number of additional gates on ibmq_almaden. Using the selection of SWAP and Bridge gate, our HA algorithm can outperform SABRE on circuits with different sizes. For medium circuits, HA gives similar results as SABRE and an improvement from SABRE for only one circuit among the eight circuits tested. For large circuits, HA outperforms SABRE and consistently reduces the number of additional gates by 28% on average. Table 4 shows the number of additional gates on ibmq_tokyo when comparing our HA algorithm with SABRE and DL. DL outperforms SABRE and our HA algorithm can further reduce the number of additional gates by 14% on average. SABRE and DL only provide their runtime on ibmq_tokyo, the difference between runtime of the three algorithms is shown in Table 4. Note that, DL is written in C++ and tested on a normal personal computer. SABRE is written in Python and tested on a server with 2 Intel Xeon E5-2680 CPUs (48 logical cores) and 378GB memory. Since there is an intrinsic speed difference between C++ and Python as well as the different devices used, the runtime data in this table are for reference rather than for comparison.

## 5 DISCUSSION

### 5.1 Design guideline

Given the current available NISQ hardware, it is important to adapt quantum programs to execute on such hardware while taking into account their physical constraints and limitations (noisy operations, number of qubits and gates). Here, we list several guidelines that can help a programmer to design quantum circuits that comply on given quantum hardware.

- Check the topology and the calibration data of the device targeted. Try to map the most used qubit of the mapped circuit to the physical qubit that has the strongest coupling connection.

- Try to apply a CNOT gate on qubits that are directly connected and with a reliable (i.e. low error rate) interconnect, so that no more additional gates are needed, and the overall circuit fidelity is improved.

- If a CNOT gate cannot be applied on two-adjacent qubits, try to apply on two qubits whose distance is two on the coupling graph. In such situation, one can select between a SWAP and Bridge gate to execute the CNOT gate. Also, the number of additional gates will be reduced.

### 5.2 Future work

In this work, we present an efficient hardware-aware mapping algorithm based on heuristic search. For future studies, we find

that the following potential research directions can be explored. First, our HA algorithm only takes into consideration the calibration data, which includes the gate error and the execution time. However, other physical constraints, such as crosstalk error may be included to take into account crosstalk coupling between interconnects. Secondly we would like to investigate the adaptation of such a mapping algorithm to a multi-programming mechanism as introduced in [12]. Executing multiple quantum circuits on the same chip allows us to use more efficiently hardware resources but may decrease the fidelity of the quantum operations due to unwanted interactions. Finally, we find it relevant to investigate mapping algorithms for specific use cases such as quantum circuits constructed for quantum chemistry computations with VQE [32] or to solve linear systems with the VQLS algorithm [7].

## 6 CONCLUSION

The quantum computers are now in the NISQ era. There's a gap between the design and execution of a quantum circuit in NISQ hardware. In this paper, we present a hardware-aware heuristic for qubit mapping problem that adapts the quantum circuit to the quantum hardware. We design a mapping transition algorithm that uses calibration data and selects from either a SWAP or Bridge gate for qubit movement. Experimental results show that our algorithm can outperform state-of-the-art algorithms in terms of the number of additional gates, fidelity and execution time. Our algorithm is evaluated on IBM quantum devices, but should be general enough to be used on quantum devices from other vendors as well.

## SUPPLEMENTARY INFORMATION

Authors have made available the source code and it can be found at the following link: https://github.com/peachnuts/HA.

## REFERENCES

[1] IBMQ backends information. https://github.com/Qiskit/ibmq-device-information, 2019. Accessed: 2019-09-18.

[2] Mohammad Alfailakawi, Imtiaz Ahmad, and Suha Hamdan. Lnn reversible circuit realization using fast harmony search based heuristic. In *Asia-Pacific Conference on Computer Science and Electrical Engineering*, 11 2014.

[3] Abdullah Ash-Saki, Mahabubul Alam, and Swaroop Ghosh. Qure: Qubit re-allocation in noisy intermediate-scale quantum computers. In *Proceedings of the 56th Annual Design Automation Conference 2019*, pages 1–6, 2019. doi: https://doi.org/10.1145/3316781.3317888.

[4] Debjyoti Bhattacharjee and Anupam Chattopadhyay. Depth-optimal quantum circuit placement for arbitrary topologies, 2017. arXiv : https://arxiv.org/abs/1703.08540.

[5] Debjyoti Bhattacharjee, Abdullah Ash Saki, Mahabubul Alam, Anupam Chattopadhyay, and Swaroop Ghosh. Muqut: Multi-constraint quantum circuit mapping on NISQ computers. In *38th IEEE/ACM International Conference on Computer-Aided Design, ICCAD 2019*, page 8942132. Institute of Electrical and Electronics Engineers Inc., 2019. doi: https://doi.org/10.1109/ICCAD45719.2019.8942132.

[6] Kyle E. C. Booth, Minh Do, J. Christopher Beck, Eleanor Rieffel, Davide Venturelli, and Jeremy Frank. Comparing and integrating constraint programming and temporal planning for quantum circuit compilation. In *International Conference on Automated Planning and Scheduling*, pages 366–374, 2018. arXiv: https://arxiv.org/abs/1803.06775.

(a) Execution time

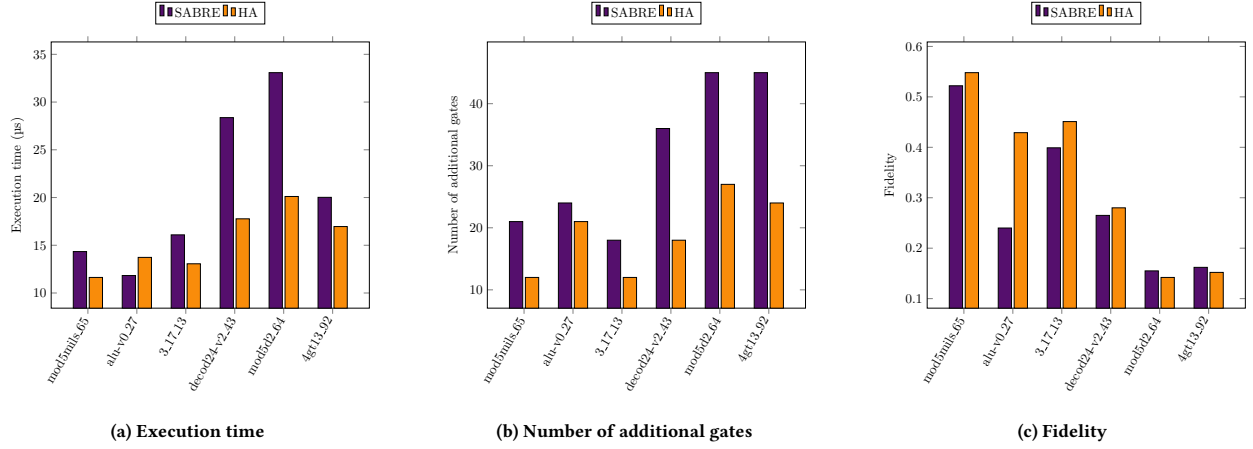(b) Number of additional gates

(c) Fidelity

**Figure 10: Comparison of execution time, number of additional gates and fidelity on ibmq_valencia. HA has been used with $\alpha_1 = 0.5$, $\alpha_2 = 0$ and $\alpha_3 = 0.5$.**

**Table 3: Number of additional gates on ibmq_almaden for large circuits. HA has been used with $\alpha_1 = 1$, $\alpha_2 = 0$ and $\alpha_3 = 0$.**

| Original Circuit | | | | SABRE | | HA | | | Comparison | |
|---|---|---|---|---|---|---|---|---|---|---|
| type | name | $n$ | $g_{all}$ | $g$ | $g_{min}$ | $g$ | $g_{min}$ | $t$ | $\Delta g\%$ | $\Delta g_{min}\%$ |
| medium | qaoa | 6 | 270 | 30 | 27 | 30 | 27 | 0.008 | 0 | 0 |
| medium | ising_model_10 | 10 | 480 | 0 | 0 | 0 | 0 | 0.02 | 0 | 0 |
| medium | ising_model_13 | 13 | 633 | 0 | 0 | 0 | 0 | 0.03 | 0 | 0 |
| medium | ising_model_16 | 16 | 786 | 3 | 0 | 9 | 0 | 0.10 | -200 | 0 |
| medium | qft_10 | 10 | 200 | 93 | 81 | 66 | 42 | 0.04 | 29 | 48.1 |
| medium | qft_13 | 13 | 403 | 192 | 177 | 195 | 171 | 0.07 | -1.6 | 3.4 |
| medium | qft_16 | 16 | 512 | 425 | 372 | 450 | 375 | 0.24 | -5.9 | -0.8 |
| large | adr4_197 | 13 | 3439 | 2973 | 2856 | 2136 | 2004 | 2.13 | 28.2 | 29.8 |
| large | radd_250 | 13 | 3213 | 2742 | 2655 | 2040 | 1926 | 1.62 | 25.6 | 27.5 |
| large | z4_268 | 11 | 3073 | 2628 | 2559 | 1872 | 1815 | 1.44 | 28.8 | 29.1 |
| large | sym6_145 | 14 | 3888 | 3024 | 2982 | 2022 | 1965 | 2.18 | 33.1 | 34.1 |
| large | misex1_241 | 15 | 4813 | 3999 | 3831 | 2892 | 2630 | 3.04 | 27.7 | 31.3 |
| large | rd73_252 | 10 | 5321 | 4539 | 4428 | 3261 | 3090 | 3.73 | 28.2 | 30.2 |
| large | cycle10_2_110 | 12 | 6050 | 5127 | 5043 | 3795 | 3576 | 4.87 | 26 | 29.1 |
| large | square_root_7 | 15 | 7630 | 6477 | 6324 | 4851 | 4707 | 7.00 | 25.1 | 25.6 |
| large | sqn_258 | 10 | 10223 | 8679 | 8580 | 6012 | 5736 | 13.92 | 30.7 | 33.1 |
| large | rd84_253 | 12 | 13658 | 11889 | 11673 | 8721 | 8574 | 24.54 | 26.6 | 26.5 |
| large | co14_215 | 15 | 17936 | 16710 | 16368 | 13071 | 12426 | 37.81 | 21.8 | 24.1 |
| large | sym9_193 | 10 | 34881 | 30558 | 30027 | 21900 | 21168 | 160.19 | 28.3 | 29.5 |
| large | 9symml_195 | 11 | 34881 | 30471 | 30129 | 21949 | 21168 | 151.84 | 28 | 29.7 |

**n**: number of qubits. **$g_{all}$**: total number of gates. **g**: average number of additional gates. **$g_{min}$**: minimum number of additional gates. **t**: runtime in seconds. **$\Delta g$**: comparison of average number of additional gates between HA and SABRE. **$\Delta g_{min}$**: comparison of minimum number of additional gates between HA and SABRE.

**Table 4: Number of additional gates on ibmq_tokyo for large circuits. HA has been used with $\alpha_1 = 1$, $\alpha_2 = 0$ and $\alpha_3 = 0$.**

| Original Circuit | | | | SABRE | | DL | | HA | | Comparison |
|---|---|---|---|---|---|---|---|---|---|---|
| type | name | $n$ | $g_{all}$ | $g$ | $t$ | $g$ | $t$ | $g$ | $t$ | $\Delta g\%$ |
| medium | ising_model_10 | 10 | 480 | 0 | 0.004 | 0 | 0 | 0 | 0.005 | 0 |
| medium | ising_model_13 | 13 | 633 | 0 | 0.007 | 0 | 0 | 0 | 0.01 | 0 |
| medium | ising_model_16 | 16 | 786 | 0 | 0.01 | 0 | 0 | 0 | 0.02 | 0 |
| medium | qft_10 | 10 | 200 | 54 | 0.103 | 39 | 0.015 | 36 | 0.015 | 7.7 |
| medium | qft_13 | 13 | 403 | 93 | 0.036 | 96 | 0.031 | 78 | 0.043 | 18.8 |
| medium | qft_16 | 16 | 512 | 186 | 0.084 | 192 | 0.062 | 174 | 0.09 | 9.4 |
| large | adr4_197 | 13 | 3439 | 1614 | 0.49 | 1224 | 0.218 | 882 | 1.41 | 27.9 |
| large | radd_250 | 13 | 3213 | 1275 | 0.48 | 1047 | 0.186 | 840 | 1.24 | 19.8 |
| large | z4_268 | 11 | 3073 | 1365 | 0.44 | 855 | 0.202 | 801 | 1.13 | 6.3 |
| large | sym6_145 | 14 | 3888 | 1272 | 0.56 | 1017 | 0.202 | 786 | 1.71 | 22.7 |
| large | misex1_241 | 15 | 4813 | 1251 | 0.89 | 1098 | 0.249 | 942 | 2.57 | 14.2 |
| large | rd73_252 | 10 | 5321 | 2133 | 0.94 | 2193 | 0.343 | 1635 | 3.19 | 25.4 |
| large | cycle10_2_110 | 12 | 6050 | 2622 | 1.35 | 1968 | 0.348 | 1719 | 4.02 | 12.7 |
| large | square_root_7 | 15 | 7630 | 2598 | 1.5 | 1788 | 0.406 | 828 | 5.66 | 53.7 |
| large | sqn_258 | 10 | 10223 | 4344 | 3.52 | 3057 | 0.563 | 2712 | 11.7 | 11.3 |
| large | rd84_253 | 12 | 13658 | 6147 | 5.39 | 5697 | 0.892 | 3843 | 21.8 | 32.5 |
| large | co14_215 | 15 | 17936 | 8982 | 9.51 | 5061 | 1.062 | 6429 | 36 | -27 |
| large | sym9_193 | 10 | 34881 | 16653 | 30.17 | 13746 | 2.091 | 11553 | 138.3 | 16 |

**n**: number of qubits. **g_all**: total number of gates. **g**: minimum number of additional gates. **t**: runtime in seconds. **Δg**: comparison of minimum number of additional gates between HA and DL.

**Table 5: Comparison of number of additional gates and fidelity on ibmq_valencia. HA has been used with $\alpha_1 = 0.5$, $\alpha_2 = 0.5$ and $\alpha_3 = 0$.**

| Original Circuit | | | SABRE | | | | HA + SABRE | | | | | HA + HSA | | | | Qiskit | | N-A | | Comparison | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| name | $n$ | $g_{all}$ | $g$ | $g_{min}$ | $S$ | $S_{max}$ | $g$ | $g_{min}$ | $S$ | $S_{max}$ | $t$ | $g$ | $g_{min}$ | $S$ | $S_{max}$ | $g$ | $S$ | $g$ | $S$ | $\Delta g\%$ | $\Delta g_{min}\%$ | $\Delta S\%$ | $\Delta S_{max}\%$ |
| BV5 | 5 | 15 | 3 | 3 | 0.576 | 0.639 | 3 | 3 | 0.612 | 0.639 | 0 | 3 | 3 | 0.581 | 0.63 | 12 | 0.456 | 3 | 0.56 | 0 | 0 | 6.3 | 0 |
| mod5mils_65 | 5 | 35 | 21 | 21 | 0.495 | 0.515 | 12 | 12 | 0.525 | 0.559 | 0.003 | 12 | 12 | 0.53 | 0.559 | 27 | 0.275 | 27 | 0.443 | 42.9 | 42.9 | 6.1 | 8.5 |
| alu-v0_27 | 5 | 36 | 24 | 24 | 0.322 | 0.329 | 18 | 18 | 0.437 | 0.437 | 0.002 | 18 | 18 | 0.384 | 0.431 | 24 | 0.335 | 24 | 0.319 | 25 | 25 | 35.7 | 32.8 |
| 3_17_13 | 3 | 36 | 18 | 18 | 0.43 | 0.476 | 12 | 12 | 0.503 | 0.546 | 0.004 | 12 | 12 | 0.463 | 0.542 | 36 | 0.458 | 21 | 0.354 | 33.3 | 33.3 | 17 | 14.7 |
| alu-v1_28 | 5 | 37 | 24 | 24 | 0.225 | 0.233 | 18 | 18 | 0.342 | 0.384 | 0.004 | 18 | 18 | 0.269 | 0.384 | 39 | 0.178 | 27 | 0.192 | 25 | 25 | 52 | 64.8 |
| decod24-v2_43 | 4 | 52 | 36 | 36 | 0.262 | 0.396 | 18 | 18 | 0.307 | 0.37 | 0.004 | 18 | 18 | 0.303 | 0.372 | 36 | 0.07 | 36 | 0.213 | 50 | 50 | 17.2 | -6.6 |
| mod5d2_64 | 5 | 53 | 45 | 45 | 0.14 | 0.208 | 24 | 24 | 0.199 | 0.207 | 0.005 | 24 | 24 | 0.194 | 0.207 | 42 | 0.171 | 48 | 0.125 | 46.7 | 46.7 | 42.1 | -0.4 |
| 4gt13_92 | 5 | 66 | 45 | 45 | 0.171 | 0.191 | 24 | 24 | 0.194 | 0.206 | 0.006 | 24 | 24 | 0.199 | 0.22 | 69 | 0.154 | 48 | 0.18 | 46.7 | 46.7 | 13.5 | 7.9 |
| ising | 5 | 90 | 24 | 24 | 0.133 | 0.145 | 24 | 24 | 0.134 | 0.141 | 0.007 | 24 | 24 | 0.137 | 0.143 | 60 | 0.113 | 33 | 0.1 | 0 | 0 | 0.8 | -2.8 |

**n**: number of qubits. **g_all**: total number of gates. **g**: average number of additional gates. **g_min**: minimum number of additional gates. **S**: mean of success rate. **S_max**: maximum of success rate. **Δg**: comparison of average number of additional gates between HA+SABRE and SABRE. **Δg_min**: comparison of minimum number of additional gates between HA+SABRE and SABRE. **ΔS**: comparison of mean of success rate between HA+SABRE and SABRE. **ΔS_max**: comparison of maximum of success rate between HA+SABRE and SABRE. **t**: runtime of HA+SABRE in seconds.

[7] Carlos Bravo-Prieto, Ryan LaRose, M. Cerezo, Yigit Subasi, Lukasz Cincio, and Patrick J. Coles. Variational quantum linear solver: A hybrid algorithm for linear systems. 09 2019. arXiv: https://arxiv.org/abs/1909.05820v2.

[8] McKay David C, Thomas Alexander, Luciano Bello, Michael J Biercuk, Lev Bishop, Jiayin Chen, Jerry M Chow, Antonio D Córcoles, Daniel Egger, Stefan Filipp, et al. Qiskit backend specifications for OpenQASM and openpulse experiments. *arXiv preprint arXiv:1809.03452*, 2018. arXiv: https://arxiv.org/abs/1809.03452.

[9] Yudong Cao, Jonathan Romero, Jonathan P Olson, Matthias Degroote, Peter D Johnson, Mária Kieferová, Ian D Kivlichan, Tim Menke, Borja Peropadre, Nicolas PD Sawaya, et al. Quantum chemistry in the age of quantum computing. *Chemical reviews*, 119(19):10856–10915, 2019. doi: https://doi.org/10.1021/acs.chemrev.8b00803.

[10] Alexander Cowtan, Silas Dilkes, Ross Duncan, Alexandre Krajenbrink, Will Simmons, and Seyon Sivarajah. On the Qubit Routing Problem. In Wim van Dam and Laura Mancinska, editors, *14th Conference on the Theory of Quantum Computation, Communication and Cryptography (TQC 2019)*, volume 135 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 5:1–5:32, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi: https://doi.org/10.4230/LIPIcs.TQC.2019.5.

[11] Andrew W Cross, Lev S Bishop, John A Smolin, and Jay M Gambetta. Open quantum assembly language. *arXiv preprint arXiv:1707.03429*, 2017. arXiv: https://arxiv.org/abs/1707.03429.

[12] Poulami Das, Swamit S Tannu, Prashant J Nair, and Moinuddin Qureshi. A case for multi-programming quantum computers. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, pages 291–303, 2019. doi: https://doi.org/10.1145/3352460.3358287.

[13] Alexandre A. A. de Almeida, Gerhard W. Dueck, and Alexandre C. R. da Silva. Finding optimal qubit permutations for ibmâĂŹs quantum computer architectures. In *Proceedings of the 32nd Symposium on Integrated Circuits and Systems Design*, SBCCI âĂŹ19, New York, NY, USA, 2019. Association for Computing

**Table 6: Comparison of number of additional gates and fidelity on ibmq_almaden. HA has been used with $\alpha_1 = 0.5$, $\alpha_2 = 0.5$ and $\alpha_3 = 0$.**

| Original Circuit | | | SABRE | | | | HA + SABRE | | | | | HA + HSA | | | | Qiskit | | N-A | | Comparison | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| name | $n$ | $g_{all}$ | $g$ | $g_{min}$ | $S$ | $S_{max}$ | $g$ | $g_{min}$ | $S$ | $S_{max}$ | $t$ | $g$ | $g_{min}$ | $S$ | $S_{max}$ | $g$ | $S$ | $g$ | $S$ | $\Delta g\%$ | $\Delta g_{min}\%$ | $\Delta S\%$ | $\Delta S_{max}\%$ |
| BV5 | 5 | 15 | 3 | 3 | 0.436 | 0.624 | 3 | 3 | 0.497 | 0.651 | 0.002 | 7 | 6 | 0.318 | 0.508 | 24 | 0.04 | 6 | 0.37 | 0 | 0 | 14 | 4.3 |
| mod5mils_65 | 5 | 35 | 21 | 21 | 0.315 | 0.47 | 12 | 12 | 0.383 | 0.481 | 0.003 | 19 | 15 | 0.268 | 0.439 | 54 | 0.107 | 33 | 0.214 | 42.9 | 42.9 | 21.6 | 2.3 |
| alu-v0_27 | 5 | 36 | 21 | 21 | 0.276 | 0.413 | 15 | 15 | 0.3 | 0.483 | 0.002 | 26 | 19 | 0.265 | 0.408 | 36 | 0.127 | 36 | 0.139 | 28.6 | 28.6 | 8.7 | 16.9 |
| 3_17_13 | 3 | 36 | 18 | 18 | 0.333 | 0.469 | 12 | 12 | 0.395 | 0.519 | 0.002 | 12 | 12 | 0.35 | 0.502 | 33 | 0.216 | 27 | 0.207 | 33.3 | 33.3 | 18.6 | 10.7 |
| alu-v1_28 | 5 | 37 | 24 | 24 | 0.25 | 0.359 | 15 | 15 | 0.391 | 0.478 | 0.002 | 21 | 21 | 0.27 | 0.408 | 48 | 0.054 | 30 | 0.087 | 37.5 | 37.5 | 56.4 | 33.1 |
| decod24-v2_43 | 4 | 52 | 36 | 36 | 0.199 | 0.334 | 18 | 18 | 0.284 | 0.401 | 0.006 | 20 | 18 | 0.235 | 0.387 | 54 | 0.076 | 39 | 0.145 | 50 | 50 | 42.7 | 20.1 |
| mod5d2_64 | 5 | 53 | 45 | 45 | 0.132 | 0.198 | 24 | 24 | 0.16 | 0.266 | 0.003 | 33 | 33 | 0.15 | 0.263 | 54 | 0.073 | 48 | 0.056 | 46.7 | 46.7 | 21.2 | 34.3 |
| 4gt13_92 | 5 | 66 | 45 | 45 | 0.13 | 0.249 | 24 | 24 | 0.145 | 0.312 | 0.007 | 32 | 27 | 0.165 | 0.347 | 99 | 0.061 | 66 | 0.106 | 46.7 | 46.7 | 11.5 | 25.3 |
| ising | 5 | 90 | 24 | 24 | 0.115 | 0.177 | 24 | 24 | 0.133 | 0.191 | 0.01 | 36 | 30 | 0.121 | 0.235 | 51 | 0.07 | 33 | 0.054 | 0 | 0 | 15.7 | 7.9 |

**n**: number of qubits. **g_all**: total number of gates. **g**: average number of additional gates. **g_min**: minimum number of additional gates. **S**: mean of success rate. **S_max**: maximum of success rate. **Δg**: comparison of average number of additional gates between HA+SABRE and SABRE. **Δg_min**: comparison of minimum number of additional gates between HA+SABRE and SABRE. **ΔS**: comparison of mean of success rate between HA+SABRE and SABRE. **ΔS_max**: comparison of maximum of success rate between HA+SABRE and SABRE. **t**: runtime of HA+SABRE in seconds.

Machinery. doi: https://doi.org/10.1145/3338852.3339829.

[14] Frank Arute et. al. Quantum supremacy using a programmable superconducting processor. *Nature*, 574:505–510, 10 2019. doi: https://doi.org/10.1038/s41586-019-1666-5.

[15] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. A quantum approximate optimization algorithm. 11 2014. arXiv: https://arxiv.org/abs/1411.4028v1.

[16] Will Finigan, Michael Cubeddu, Thomas Lively, Johannes Flick, and Prineha Narang. Qubit allocation for noisy intermediate-scale quantum computers. *arXiv preprint arXiv:1810.08291*, 2018. arXiv: https://arxiv.org/abs/1810.08291.

[17] Robert W Floyd. Algorithm 97: shortest path. *Communications of the ACM*, 5(6):345, 1962. doi: https://doi.org/10.1145/367766.368168.

[18] András Gilyén, Yuan Su, Guang Hao Low, and Nathan Wiebe. Quantum singular value transformation and beyond: exponential improvements for quantum matrix arithmetics. 06 2018. doi: https://doi.org/10.1145/3313276.3316366.

[19] Gian Giacomo Guerreschi. Scheduler of quantum circuits based on dynamical pattern improvement and its application to hardware design. *arXiv e-prints*, page arXiv:1912.00035, November 2019. arXiv: https://arxiv.org/abs/1912.00035.

[20] Gian Giacomo Guerreschi and Jongsoo Park. Two-step approach to scheduling quantum circuits. *Quantum Science and Technology*, 3(4):045003, jul 2018. doi: https://doi.org/10.1088/2058-9565/aacf0b.

[21] Aram W. Harrow, Avinatan Hassidim, and Seth Lloyd. Quantum algorithm for linear systems of equations. *Physical Review Letters*, 103, 10 2009. doi: https://doi.org/10.1103/PhysRevLett.103.150502.

[22] Hsin-Yuan Huang, Kishor Bharti, and Patrick Rebentrost. Near-term quantum algorithms for linear systems of equations. 09 2019. arXiv: https://arxiv.org/abs/1909.07344v1.

[23] Toshinari Itoko, Rudy Raymond, Takashi Imamichi, and Atsushi Matsuo. Optimization of quantum circuit mapping using gate transformation and commutation. *Integration*, 70:43–50, 2020. doi: https://doi.org/10.1016/j.vlsi.2019.10.004.

[24] Iordanis Kerenidis and Anupam Prakash. A quantum interior point method for lps and sdps. 08 2018. arXiv: https://arxiv.org/abs/1808.09266.

[25] Iordanis Kerenidis and Anupam Prakash. Quantum gradient descent for linear systems and least squares. *Phys. Rev. A*, 101:022316, Feb 2020. doi: https://doi.org/10.1103/PhysRevA.101.022316.

[26] Abhoy Kole, Kamalika Datta, and Indranil Sengupta. A heuristic for linear nearest neighbor realization of quantum circuits by swap gate insertion using *n*-gate lookahead. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 6(1):62–72, 2016. doi: https://doi.org/10.1109/JETCAS.2016.2528720.

[27] Lingling Lao, Hans van Someren, Imran Ashraf, and Carmen G. Almudever. Mapping of quantum circuits onto NISQ superconducting processors. *arXiv e-prints*, August 2019. arXiv: https://arxiv.org/abs/1908.04226v1.

[28] Ang Li and Sriram Krishnamoorthy. QASMBench: A low-level QASM benchmark suite for NISQ evaluation and simulation. *arXiv preprint arXiv:2005.13018*, 2020. arXiv: https://arxiv.org/abs/2005.13018.

[29] Gushu Li, Yufei Ding, and Yuan Xie. Tackling the qubit mapping problem for NISQ-era quantum devices. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 1001–1014, 2019. doi: https://doi.org/10.1145/3297858.3304023.

[30] Prakash Murali, Jonathan M. Baker, Ali Javadi-Abhari, Frederic T. Chong, and Margaret Martonosi. Noise-adaptive compiler mappings for noisy intermediate-scale quantum computers. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS âĂŹ19, page 1015âĂŞ1029, New York, NY, USA, 2019. Association for Computing Machinery. doi: https://doi.org/10.1145/3297858.3304075.

[31] Prakash Murali, Norbert Matthias Linke, Margaret Martonosi, Ali Javadi Abhari, Nhung Hong Nguyen, and Cinthia Huerta Alderete. Full-stack, real-system quantum computer studies: Architectural comparisons and design insights, 2019. doi: https://doi.org/10.1145/3307650.3322273.

[32] Alberto Peruzzo, Jarrod McClean, Peter Shadbolt, Man-Hong Yung, Xiao-Qi Zhou, Peter J. Love, Alán Aspuru-Guzik, and Jeremy L. O'Brien. A variational eigenvalue solver on a quantum processor. 04 2013. doi: https://doi.org/10.1038/ncomms5213.

[33] John Preskill. Quantum computing in the NISQ era and beyond. 01 2018. doi: https://doi.org/10.22331/q-2018-08-06-79.

[34] Mehdi Saeedi, Robert Wille, and Rolf Drechsler. Synthesis of quantum circuits for linear nearest neighbor architectures. *Quantum Information Processing*, 10(3):355–377, 2011. doi: https://doi.org/10.1007/s11128-010-0201-2.

[35] Changpeng Shao and Hua Xiang. Row and column iteration methods to solve linear systems on a quantum computer. *Phys. Rev. A*, 101:022322, 02 2020. doi: https://doi.org/10.1103/PhysRevA.101.022322.

[36] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J. Sci. Statist. Comput. 26 (1997) 1484*, 09 1995. doi: https://doi.org/10.1137/S0097539795293172.

[37] Ritu Ranjan Shrivastwa, Kamalika Datta, and Indranil Sengupta. Fast qubit placement in 2d architecture using nearest neighbor realization. In *2015 ieee international symposium on nanoelectronic and information systems*, pages 95–100. IEEE, 2015. doi: https://doi.org/10.1109/iNIS.2015.59.

[38] Marcos Yukio Siraichi, Vinícius Fernandes dos Santos, Sylvain Collange, and Fernando Magno Quintao Pereira. Qubit allocation. In *Proceedings of the 2018 International Symposium on Code Generation and Optimization*, CGO 2018, page 113âĂŞ125, New York, NY, USA, 2018. Association for Computing Machinery. doi: https://doi.org/10.1145/3168822.

[39] Swamit S Tannu and Moinuddin K Qureshi. Not all qubits are created equal: a case for variability-aware policies for NISQ-era quantum computers. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 987–999, 2019. doi: https://doi.org/10.1145/3297858.3304007.

[40] Davide Venturelli, Minh Do, Bryan O'Gorman, Jeremy Frank, Eleanor Rieffel, Kyle EC Booth, Thanh Nguyen, Parvathi Narayan, and Sasha Nanda. Quantum circuit compilation : An emerging application for automated reasoning. 2019.

[41] R. Wille, D. Große, L. Teuber, G. W. Dueck, and R. Drechsler. RevLib: An online resource for reversible functions and reversible circuits. In *Int'l Symp. on -Valued Logic*, pages 220–225, 2008. RevLib is available at http://www.revlib.org.

[42] Robert Wille, Oliver Keszocze, Marcel Walter, Patrick Rohrs, Anupam Chattopadhyay, and Rolf Drechsler. Look-ahead schemes for nearest neighbor optimization of 1d and 2d quantum circuits. In *2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 292–297. IEEE, 2016. doi: https://doi.org/10.1109/ASPDAC.2016.7428026.

[43] Xiaosi Xu, Jinzhao Sun, Suguru Endo, Ying Li, Simon C. Benjamin, and Xiao Yuan. Variational algorithms for linear algebra. 09 2019. arXiv: https://arxiv.org/abs/1909.03898.

[44] Xiangzhen Zhou, Sanjiang Li, and Yuan Feng. Quantum circuit transformation based on simulated annealing and heuristic search. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2020. doi: https://doi.org/10.1109/TCAD.2020.2969647.

[45] P. Zhu, Z. Guan, and X. Cheng. A dynamic look-ahead heuristic for the qubit mapping problem of NISQ computers. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2020. doi: https://doi.org/10.1109/TCAD.2020.2970594.

[46] Alwin Zulehner, Alexandru Paler, and Robert Wille. An efficient methodology for mapping quantum circuits to the ibm qx architectures. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 38(7):1226–1236, 2018. doi: https://doi.org/10.1109/TCAD.2018.2846658.