Qubit Mapping and Routing via MaxSAT

Abtin Molavi, Amanda Xu, Martin Diges, Lauren Pick, Swamit Tannu, Aws Albarghouthi
University of Wisconsin-Madison, Madison, WI, USA
{amolavi, axu44, mdiges, lpick2, stannu, albarghouthi}@wisc.edu

***Abstract*—Near-term quantum computers will operate in a noisy environment, without error correction. A critical problem for near-term quantum computing is laying out a logical circuit onto a physical device with limited connectivity between qubits. This is known as the *qubit mapping and routing* (QMR) problem, an intractable combinatorial problem. It is important to solve QMR as optimally as possible to reduce the amount of added noise, which may render a quantum computation useless. In this paper, we present a novel approach for optimally solving the QMR problem via a reduction to *maximum satisfiability* (MAXSAT). Additionally, we present two novel relaxation ideas that shrink the size of the MAXSAT constraints by exploiting the structure of a quantum circuit. Our thorough empirical evaluation demonstrates (1) the scalability of our approach compared to state-of-the-art optimal QMR techniques (*solves more than 3x benchmarks with 40x speedup*), (2) the significant cost reduction compared to state-of-the-art heuristic approaches (*an average of ∼5x swap reduction*), and (3) the power of our proposed constraint relaxations.**

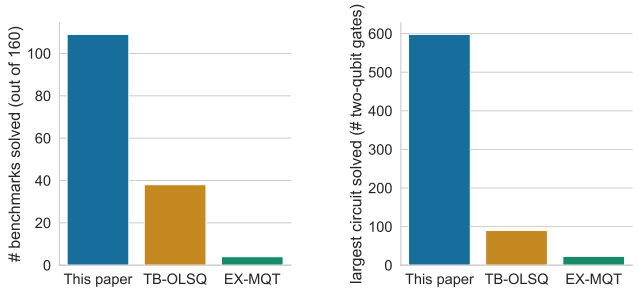
***Index Terms*—quantum computing, qubit mapping**

I. Introduction

Quantum computers enable efficient simulation of quantum mechanical phenomena, and therefore open up the door to advances in quantum physics, chemistry, material design, optimization, machine learning, and beyond. Unfortunately, near-term quantum computers face significant reliability challenges as quantum hardware is highly error-prone: quantum bits (qubits) used for computation are sensitive to environmental noise. Furthermore, implementing *quantum error correction* [1] to detect and correct hardware errors requires thousands of physical qubits, and therefore is unlikely to become viable soon. In the meantime, near-term quantum computers with several dozens of qubits are expected to operate in a noisy environment without any error correction using a model of computation called *noisy intermediate-scale quantum* (NISQ) computing [2].

A critical problem in NISQ computing is laying out a logical circuit onto a physical device with limited connectivity between qubits. This is known as the *qubit mapping and routing* (QMR) problem. Specifically, we can only apply two-qubit gates on physically adjacent qubits, so we need to move (*route*) qubits to physically adjacent locations. Qubit routing is a noisy process that can be detrimental to successful execution. Thus, our goal is to lay out the circuit in such a way that minimizes the required routing.

Solving QMR optimally is known to be NP-hard [3]. Thus, a majority of the proposed techniques have been heuristic in nature, producing suboptimal results [4]. A small number of techniques have been proposed for solving QMR optimally, mostly by reducing the problem to optimizing an objective function subject to constraints, e.g., *integer linear programming* or *satisfiability modulo theories* [5], [6], [7]. While such *constraint-based* approaches produce optimal results with minimum noise, they have not been scalable to larger circuits.



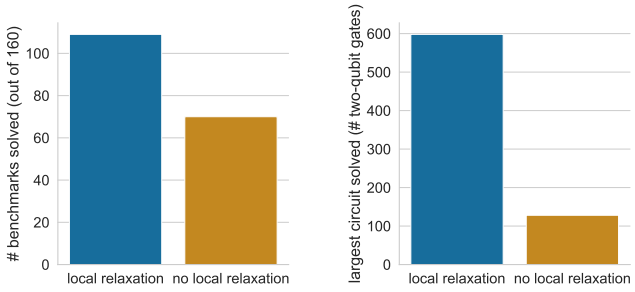
(a) Number of benchmarks solved **(b) Size of largest circuit solved**

Fig. 1: Comparison against constraint-based tools

In this paper, we propose a novel constraint-based approach that significantly advances the state of the art (see Fig. 1). We believe that scaling constraint-based approaches is an important problem for two reasons: (1) With heuristic QMR techniques, one can easily add an unacceptable amount of noise for NISQ computers, producing uninformative outputs. (2) Constraint-based techniques present an optimal baseline with which to evaluate the solution quality of heuristic algorithms, and can therefore help us understand and improve their operation.

QMR as MAXSAT. Our primary insight is that we can reduce the QMR problem to *maximum satisfiability* (MAXSAT) [8, Chapter 19]. MAXSAT is the optimization analogue of the Boolean satisfiability (SAT) problem. While SAT solving is the canonical NP-complete problem, the past two decades have witnessed impressive advances in SAT solving with industrial-grade tools applied at scale (e.g., at Amazon [9], SAT solvers are invoked millions of times daily). MAXSAT solvers are typically simple loops that repeatedly invoke a SAT solver to get better and better solutions. Compared to other approaches that use *satisfiability modulo theories* (SMT) solvers [5], [6], [7], MAXSAT solvers are lighter weight as they do not require complex theory-solver interaction. At a high level, we demonstrate that a MAXSAT approach *can* and *should* be used for solving QMR constraints.

As summarized in Fig. 1, compared to state-of-the-art constraint-based tools [5], [10], our approach can solve significantly more QMR problems (∼3x) and scale to larger



(a) Number of benchmarks solved **(b) Size of largest circuit solved**

Fig. 2: Comparison against enabling local relaxation

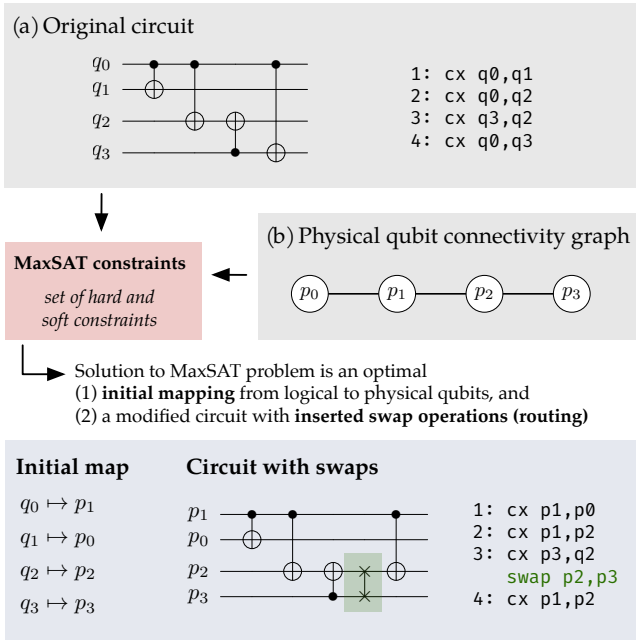


Fig. 3: Running example and overview

circuits. In addition, our approach is an *order of magnitude faster* (∼40x) than the fastest constraint-based tool. Compared to state-of-the-art heuristic-based QMR tools, our approach achieves an average of ∼3.6x to 7x reduction in the number of inserted swap operations. Further, on ∼14% of the benchmarks, our approach inserts no swap operations at all.

Sketching-like encoding. Our MAXSAT encoding is inspired by *program sketching* [11], a program-synthesis paradigm where a synthesizer automatically completes *holes* in a program. In our setting, these *holes* are the routing operations—specifically, SWAP gates that exchange the contents of two adjacent qubits. We encode every possible SWAP as a Boolean variable, where assigning the variable to `true` denotes performing a SWAP of a specific pair of adjacent physical qubits. We therefore ask the MAXSAT solver to minimize the number of SWAP variables set to `true`.

Relaxation techniques. While solving MAXSAT constraints results in an optimal QMR solution, for large circuits, the MAXSAT solver may not be able to efficiently solve the generated constraints. In this paper, we demonstrate a novel relaxation of constraint-based solutions to QMR, which we call *locally* optimal relaxation. The idea is to *slice* the circuit horizontally into a number of consecutive subcircuits and solve a set of smaller MAXSAT problems for each of them. We demonstrate that our locally optimal relaxation can scale our approach (as shown in Fig. 2) while still producing almost-optimal results (as detailed in Section VII).

Additionally, we present a relaxation for efficiently mapping *cyclic circuits*, like in *quantum approximate optimization algorithms* (QAOA) [12]. Cyclic circuits are where the same subcircuit is repeated more than once. Instead of generating one big set of constraints for the entire circuit, we solve a special set of MAXSAT constraints only for the repeated subcircuit in isolation, and then *stitch* the subcircuits back together to generate a mapping for the entire circuit. Our results demonstrate that this relaxation can make our technique scale to larger QAOA circuits.

Contributions. We summarize our contributions as follows:

- A novel constraint-based approach for optimally solving the qubit mapping and routing problem via a reduction to maximum satisfiability (MAXSAT).
- A locally optimal constraint relaxation based on circuit slicing.
- A specialized constraint relaxation for cyclic circuits, e.g., as in QAOA.
- A thorough empirical evaluation demonstrating (1) the scalability of our approach compared to state-of-the-art constraint-based techniques, (2) the significant cost reduction compared to heuristic approaches, and (3) the power of our proposed constraint relaxations.

II. An Illustrative Example

In this section, we provide background on the qubit mapping and routing (QMR) problem and walk through a running example that motivates our approach.

QMR primer. Quantum computers typically support two kinds of operations, single-qubit gates (e.g., NOT, analogous to a bit flip), and two-qubit gates (e.g., CNOT, analogous to an exclusive or). Due to various physical design constraints, NISQ-era quantum computers support two-qubit operations only between certain pairs of physical qubits as described by a *connectivity graph*. Consider, for instance, the simple connectivity graph in Fig. 3(b), which illustrates a small device with four physical qubits, $p_0, \ldots, p_3$. Edges between physical qubits denote whether we can perform two-qubit operations between them. For example, we can perform a two-qubit operation over p_0 and p_1, but not p_0 and p_2.

In order to execute a quantum circuit on a particular device, the compiler *maps* the logical qubits that appear in the circuit to appropriate physical qubits such that every two-qubit gate can be applied. Consider the circuit in Fig. 3(a); the first gate is a CNOT between logical qubits q_0 and q_1 (denoted in the

assembly code on the right as `cx q0,q1`). Therefore, the logical qubits q_0 and q_1 should be mapped to physical qubits that are adjacent in the connectivity graph, e.g., physical qubits p_3 and p_4 in the graph in Fig. 3(b).

Typically, a static initial map does not suffice and so the map has to be transformed during circuit execution to accommodate for other two-qubit gates later on in the circuit. This process, called *routing*, is achieved by inserting SWAP operations, which exchange the values of two connected qubits. For instance, suppose we want to perform a two-qubit gate on p_1 and p_3. They are not connected in the connectivity graph; therefore, we need to bring them *next to each other*. One way to do so is to swap the qubits p_2 and p_3.

Our goal is to solve the QMR problem *optimally*:

Find an initial map that requires the least routing (number of swap operations to be inserted).

Each additional gate in the circuit increases the probability of error. In particular, two-qubit gate error rates are significantly higher than one-qubit gate error rates. In addition, two-qubit gates such SWAPs have significantly longer gate latency, which can make qubits prone to decoherence errors, so minimizing the number of SWAPs is critical.

Our MAXSAT approach. Finding an optimal QMR solution is a combinatorially challenging problem; indeed, it is NP-complete. In this paper, we capitalize on the success of satisfiability (SAT) solvers for finding satisfying assignments of Boolean formulas. E.g., for a Boolean formula $a \wedge \neg b$, setting a to `true` and b to `false` is a satisfying assignment. While the satisfiability problem is the quintessential NP-complete problem, algorithmic and engineering progress in satisfiability has made SAT solving practical in many instances [8]. Since QMR is an optimization problem, we use a MAXSAT solver, which builds upon a SAT solver to find an optimal satisfying assignment.

Roughly speaking, given a circuit and a connectivity graph, we generate a set of Boolean formulas (constraints) whose optimal satisfying assignment corresponds to an initial mapping of logical to physical qubits and a set of SWAP operations to be inserted before each two-qubit gate. Specifically, the crucial bit of our encoding is that we model all possible SWAP operations as Boolean variables; for example, `swap p0,p1` would have a corresponding Boolean variable in every location it could be placed in the circuit. Then, if a Boolean variable is assigned to `true` in the solution to the MAXSAT problem, the corresponding SWAP is inserted into the circuit; otherwise, it is not.

Fig. 3 provides an example of the QMR problem. The circuit Fig. 3(a) applies two-qubit operations between the logical qubit q_0 and three other logical qubits, but every physical qubit is only connected to at most two other physical qubits. Therefore, SWAPs are needed here. It turns out that inserting a single swap is sufficient for this example. Fig. 3(bottom) shows an optimal QMR solution that can be discovered by solving the MAXSAT constraints (the inserted SWAP is highlighted in green).

A MAXSAT solver is typically implemented as a loop that queries a SAT solver for better and better solutions, until it arrives at an optimal one. Therefore, a benefit of using a MAXSAT solver is that, even for large circuits where the solver cannot efficiently find an optimal solution, the solver may be terminated early to extract the best solution found so far (if it has progressed past the first loop iteration).

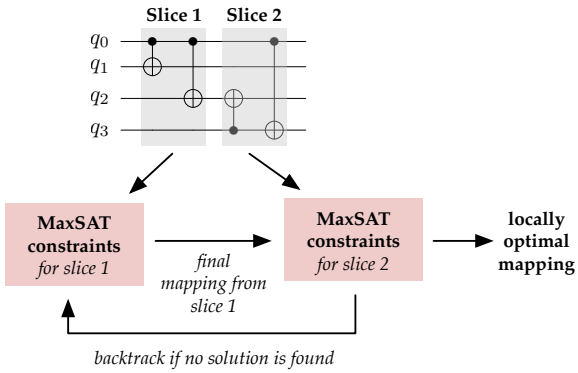


Fig. 4: Illustration of our locally optimal relaxation

Slicing and cyclic circuits. As mentioned previously, for large circuits, the MAXSAT solver may not be able to efficiently solve the generated constraints. In the worst case, the MAXSAT solver will not even find a non-optimal solution to the QMR problem in a feasible amount of time. In this paper, we demonstrate a locally optimal relaxation of the constraints, in which we slice the circuit horizontally into a number of consecutive subcircuits and solve a set of smaller MAXSAT problems for each of them. This relaxation allows us to scale our approach to larger circuits by sacrificing a guarantee of global optimality.

This idea is illustrated for our running example in Fig. 4. First we slice the circuit into two slices (it could be more, but we stick to two slices for illustration), as shown in the shaded areas. We solve the MAXSAT constraints for the first slice; this generates an optimal solution for the slice in isolation. We then take the final mapping from this solution—i.e., the mapping at the end of slice 1 after all swaps have been executed—and add it to the constraints for solving slice 2. As we describe in Section V, in some cases we need to backtrack as the solution of slice 1 may be incompatible with slice 2.

We also present an analogous idea for efficiently solving QMR for *cyclic circuits*, which consist of repeated instances of the same subcircuit. Rather than generating a monolithic set of constraints for the entire circuit at once, we instead generate and solve a special set of MAXSAT constraints only for the repeated subcircuit in isolation. We then stitch copies of the solution to generate a mapping for the entire circuit. We describe this idea in Section VI.

The two aforementioned relaxations can be easily composed by slicing subcircuits of a cyclic circuit, thus exploiting both the cyclic and slice-like structure of a circuit.

III. Qubit Mapping and Routing

We now define the qubit mapping and routing problem.

Connectivity graph. We will use $G = (Phys, Edges)$ to denote a connectivity graph between physical qubits on a quantum device, where

- $Phys = \{p_0, p_1, \ldots\}$ is the set of physical qubits and
- $Edges \subseteq Phys \times Phys$ is the set of edges connecting physical qubits.

Graph edges denote on which pairs of physical qubits we can perform two-qubit operations.

Quantum circuit. We will use C to denote a quantum circuit over logical qubits, $Logic = \{q_0, q_1, \ldots\}$. Specifically, a circuit C is a sequence of gate applications, where each gate is an operation that applies to one or two logical qubits. We will use g_k to denote the kth gate in the circuit, $g_k(q)$ to denote the application of the one-qubit gate g_k to logical qubit q, or $g_k(q, q')$ to denote the application of the two-qubit gate g_k to qubits q and q'.

Qubit map. Given a circuit C and an undirected connectivity graph G, a *qubit map* $M : Logic \to Phys$ is an injective function from logical qubits to physical qubits.

Our goal is to find a *map sequence* $\langle M_1, \ldots, M_{|C|} \rangle$, where $|C|$ is the number of gates in the circuit, such that if the kth gate in the circuit is a two-qubit gate $g_k(q, q')$, then

$$(M_k(q), M_k(q')) \in Edges,$$

i.e., logical qubits q and q' are mapped to physical qubits that are connected in the connectivity graph.

Example 1: Recall our running example. The initial map, M_1, is shown in Fig. 3(bottom). Observe that logical qubits q_0 and q_1 are mapped to adjacent physical qubits, p_1 and p_0, i.e., $(M(q_0), M(q_1)) \in Edges$.

SWAP operations. We will use $s(p, p')$ to denote the SWAP operation that swaps the physical qubits p and p'.

Suppose that in qubit map M we have $M(q) = p$ and $M(q') = p'$. Applying $s(p, p')$ in M results in a new map M' that is just like M but where $M'(q) = p'$ and $M'(q') = p$. Note that SWAP operations are only allowed on pairs of connected physical qubits, i.e., $(p, p') \in Edges$.

Optimal qubit mapping and routing (QMR). An *optimal* solution to the QMR problem is a map sequence $\langle M_1, \ldots, M_{|C|} \rangle$ that minimizes the cost of *routing* qubits between adjacent maps in the sequence; formally:

$$\min \sum_{i=1}^{|C|-1} cost(M_i, M_{i+1})$$

where $cost(M, M')$ is the smallest number of SWAP operations needed to go from M to M'.

Example 2: Continuing our running example: Only one swap operation happens, right before the fourth gate, making the total cost 1. So, $M_3 = M_2 = M_1$. Before the fourth gate, the physical qubits p_2 and p_3 are swapped, resulting in the map M_4 that is the same as M_1 except that $M_4(q_2) = p_3$ and $M_4(q_3) = p_2$.

IV. Optimal QMR via MaxSAT

In this section, we will present our approach for discovering an optimal solution of the QMR problem via a reduction to *maximum satisfiability* (MAXSAT). We begin by providing some background on MAXSAT.

A. MaxSAT Background

MAXSAT is the optimization analogue of the classical Boolean satisfiability problem, SAT. Before turning to our encoding, we will define both of these problems.

The SAT problem. In the *satisfiability* problem (SAT), we are given a Boolean formula and our goal is to find an assignment to the variables that makes the formula true—a *model* of the formula. We will use standard notation to denote Boolean operations: $\wedge$ (AND), $\vee$ (OR), $\neg$ (NOT), and $\to$ (implication).

Example 3: Consider the following formula, where a, b, c are Boolean variables:

$$(\neg a \wedge b) \to c$$

This formula is *satisfiable* because there is a model that makes it true. One such model is:

$$I = [a \mapsto \mathsf{false},\ b \mapsto \mathsf{true},\ c \mapsto \mathsf{true}]$$

Given a formula ϕ, we will use $I \models \phi$ to denote that I is a model of ϕ. If ϕ has no models, then it is *unsatisfiable*.

The MAXSAT problem. In the MAXSAT problem, we are given two sets of Boolean formulas: *Hard* constraints and *Soft* constraints. Our goal is to find a single assignment I that is a model of all hard constraints and as many soft constraints as possible.

Example 4: Consider the MAXSAT problem with one hard constraint and two soft ones:

$$Hard = \{\neg a \vee b\}$$
$$Soft = \{b,\ a \wedge \neg b\}$$

Since $a \wedge \neg b$ in *Soft* is the negation of $\neg a \vee b$, there is no *model* of *Hard* that is also a model of $\{a \wedge \neg b\}$. Therefore the maximum number of formulas from *Soft* that can evaluate to true is one. A solution is $I = [a \mapsto \mathsf{false}, b \mapsto \mathsf{true}]$.

B. MaxSAT Encoding of Optimal QMR

We will now present our MAXSAT encoding for optimally solving the QMR problem. Throughout, we fix a circuit C over logical qubits *Logic* and a connectivity graph $G = (Phys, Edges)$.

Our encoding will define a set of hard constraints and a set of soft constraints, $(Hard, Soft)$, constituting a MAXSAT instance. A solution to this MAXSAT instance yields an optimal solution to the QMR problem, specifically, (1) an optimal map sequence, $\langle M_1, \ldots, M_{|C|} \rangle$, and (2) a sequence of SWAPs before every two-qubit gate to perform routing. The soft constraints aim to minimize the number of inserted SWAPs.

Our encoding uses two sets of Boolean variables: the map variables, which represent the sequence of maps $\langle M_1, \ldots, M_{|C|} \rangle$,

Mapping constraints:

***Hard A*: Maps are injective functions.** For every gate g_k in the circuit, every logical qubit q, and every pair of distinct physical qubits p, p', we add the following hard constraint:

$$\mathsf{map}(q,p,k) \rightarrow \neg\mathsf{map}(q,p',k)$$

Similarly, for every g_k, every pair of distinct logical qubits, q, q', and every physical qubit p, we add the following hard constraint:

$$\mathsf{map}(q,p,k) \rightarrow \neg\mathsf{map}(q',p,k)$$

***Hard B*: Executing two-qubit gates.** For every two-qubit gate $g_k(q,q')$, we add the following hard constraint:

$$\bigvee_{(p,p')\in Edges} (\mathsf{map}(q,p,k) \wedge \mathsf{map}(q',p',k))$$

Routing constraints:

***Hard C*: Only one swap.** For the kth gate and for its ith SWAP, we add the hard constraint:

$$\bigvee_{(p,p')\in Edges'} \big(\mathsf{swap}(p,p',k,i) \wedge unique(p,p')\big)$$

where

$$unique(p,p') \triangleq \bigwedge_{(r,r')\in Edges'\setminus\{(p,p')\}} \neg\mathsf{swap}(r,r',k,i)$$

Here $Edges' = Edges \cup \{(p_0,p_0)\}$, a synthetic edge used to denote a no-op SWAP.

***Hard D*: The effect of SWAPs.** For every gate g_k and sequence S of swaps of physical qubits, $s(p_0,p_0'),\ldots,s(p_{n-1},p_{n-1}')$, we add the following hard constraint:

$$\left(\bigwedge_{0\leqslant i<n} \mathsf{swap}(p_i,p_i',i,k)\right) \rightarrow effect(S)$$

where

$$effect(S) \triangleq \bigwedge_{\substack{q\in Logic,\\ phys\in Phys}} (\mathsf{map}(q,p,k-1) \leftrightarrow \mathsf{map}(q,\pi(S,p),k))$$

Soft constraints:

***Soft*: Minimize the number of SWAPs.** For every gate k, we add the following soft constraint:

$$\mathsf{swap}(p_0,p_0,k)$$

Fig. 5: Formalization of our MAXSAT encoding

and the swap variables, which represent where SWAPs are inserted in the circuit. In what follows, we describe our constraints in a semi-formal manner with examples and refer to Fig. 5 for the complete formalization.

1) Mapping Constraints: We start by describing the constraints that specify that our map sequence is valid.

We will use the Boolean variable $\mathsf{map}(q,p,k)$ to denote that, for a logical qubit $q \in Logic$ and a physical qubit $p \in Phys$, q maps to p right before the kth gate of the circuit. In other words, if $\mathsf{map}(q,p,k)$ is assigned true, then this means that $M_k(q) = p$.

Example 5: Recall our running example from Fig. 3. The initial map, M_1, shown in Fig. 3(bottom), maps q_0 to p_1. This is represented in our encoding by assigning the variable $\mathsf{map}(q_0,p_1,1)$ to true in the solution to the MAXSAT constraints. Similarly, all other variables $\mathsf{map}(q_0,p_i,1)$, where $i \neq 1$, are set to false, because q_0 can only be mapped to a single physical qubit.

***Hard A*: Maps are injective functions.** Our first set of hard constraints (formalized in Fig. 5) specify that our map variables model injective functions. Following Example 5, such constraints ensure that we cannot set both $\mathsf{map}(q_0,p_1,1)$ and $\mathsf{map}(q_0,p_2,1)$ to true in a solution of the MAXSAT constraints. Additionally, we cannot map different logical qubits to the same physical qubit.

***Hard B*: Executing two-qubit gates.** Our second set of hard constraints specify that for each two-qubit gate in the circuit, the two logical qubits it acts on are mapped to adjacent physical qubits.

Example 6: In our running example, the first gate is a CNOT over q_0 and q_1. Therefore, we should initially map q_0 and q_1 to adjacent physical qubits. One way to satisfy this is to set $\mathsf{map}(q_0,p_0,1)$ and $\mathsf{map}(q_1,p_1,1)$ to true, since (p_0,p_1) is an edge in the connectivity graph. However, we cannot satisfy this constraint by setting $\mathsf{map}(q_0,p_0,1)$ and $\mathsf{map}(q_1,p_3,1)$ to true, since there is no edge connecting physical qubits (p_0,p_3).

2) Routing Constraints: We now describe the routing constraints. The key idea is that right before a two-qubit gate, $g(q,q')$, we want to insert a sequence of SWAPs to ensure that the two logical qubits, q and q', are mapped to adjacent physical qubits.

Suppose, for illustration, that the kth gate in the circuit is a CNOT over q_0 and q_1. Right before this CNOT, we allow our encoding to insert *up to n* SWAPs. We can think of this through the lens of *program sketching* [11], where we don't know which qubits to swap before the kth gate, so we add up to n SWAPs with unknown parameters (denoted with $\bullet$ below) and the goal of our encoding is to *discover* those parameters.

```
swap •,•
...
swap •,•
  cx q0,q1
```

Specifically, for every inserted SWAP with unknown parameters, we create a number of Boolean variables denoting every possible instantiation of the parameters. Formally, the Boolean variable $\mathsf{swap}(p,p',k,i)$ denotes that the ith SWAP inserted before gate k is over physical qubits p and p'. (If both parameters are set to the same qubit, then the SWAP is considered a no-op.)

Example 7: Suppose $n = 1$—i.e., we only allow up to 1 SWAP before a two-qubit gate—and we have a device with only two physical qubits, p_0 and p_1, with an edge between them. For the fourth gate in the circuit, we will have the following set of Boolean variables:

$$\{\mathsf{swap}(p_0, p_0, 4, 1), \mathsf{swap}(p_0, p_1, 4, 1)\}$$

If the first variable is set to true in a solution to the MAXSAT constraints, then no SWAP is inserted before the fourth gate (no-op); if the second variable is set to true, then a SWAP operation is inserted that swaps p_0 and p_1.

***Hard C*: Only one swap.** As indicated by the above example, for a specific SWAP with unknown parameters, only one of its associated Boolean variables can be set to true, since there's only one possible instantiation of its parameters. This is enforced by a standard *only-one* hard constraint [13].

***Hard D*: The effect of SWAPs.** The most involved routing constraint is encoding the effect of a sequence of SWAPs on the map sequence. Specifically, we have to encode how the inserted SWAPs transform map M_{k-1} into M_k.

We define a function $\pi(S, p)$ that specifies the effect of a sequence of SWAPs S on a physical qubit p, i.e., which qubit p gets routed to after executing the swaps in S. The following example provides a simple illustration:

Example 8: For the sequence S that just swaps p and p', we have $\pi(S, p) = p'$. For the sequence S that swaps p and p' and then p' and p'', we have $\pi(S, p) = p''$.

The final set of hard constraints (Hard D) encodes the effect of every possible sequence S of n SWAPs. While the number of sequences is exponential in n, in practice, we have experimentally found that a small constant suffices for finding optimal solutions (Section VII).

3) Soft Constraints and Optimality: We have described all the required hard constraints. Finally, we define a set of soft constraints with the goal of minimizing the number of inserted SWAPs. Informally, we want to ensure that as many SWAPs are no-ops as possible. So, we maximize the number of true Boolean variables of the form $\mathsf{swap}(p, p, k)$.

Fig. 5 fully formalizes all of the hard and soft constraints that our encoding generates. So, if we are given a circuit C and a connectivity graph G, we can use our encoding to generate a MAXSAT instance $(Hard, Soft)$ whose solution results in an optimal QMR solution.

Given a model $I \models (Hard, Soft)$, we can extract a valid map sequence from the assignments of the Boolean variables of the form $\mathsf{map}(q, p, k)$ by setting $M_k(q) = p$ exactly when I assigns $\mathsf{map}(q, p, k)$ to true.

This following theorem states optimality of our solutions:

Theorem 1: Let I be a solution for $(Hard, Soft)$. Let $\langle M_1, \ldots, M_{|C|} \rangle$ be the map sequence corresponding to I as described above. Then, $\langle M_1, \ldots, M_{|C|} \rangle$ is an optimal solution of the QMR problem (as per Section III).

In the theorem above, we make the assumption that n (the number of SWAPs allowed before each CNOT) is set to the *diameter* of the connectivity graph. This ensures that we can always bring any two qubits into adjacent positions.

Encoding size. If n, the number of SWAPs allowed before each CNOT, is held constant (see Section VII), the MAXSAT encoding from Section IV scales polynomially with the size of the input circuit and architecture. In particular, a naive implementation requires $O(|C| \cdot |Edges|)$ variables and $O(|Phys|^2 \cdot |Logic| \cdot |C|)$ constraints (Hard A is the dominating term). However, using a standard "only-one" encoding [13], we need only $O(|Phys| \cdot |Logic| \cdot |C|)$ constraints. This is a more compact representation than EX-MQT [6] and matches the asymptotic behavior of a subsequent SMT approach, TB-OLSQ [5]. While the number of constraints is roughly the same as TB-OLSQ, our sketch-based view allows us to eschew the use of integer arithmetic, eliminating the expensive theory-lemma generation of an SMT solver.

V. A Locally Optimal Relaxation

Solving the MAXSAT encoding presented in the previous section results in an optimal QMR solution. However, this can be expensive in practice due to the complexity of MAXSAT. In this section, we will present a relaxation that produces *locally* optimal solutions. Specifically, our approach *slices* the circuit into a number of subcircuits and solves a MAXSAT problem for each of them in sequence. The result is that we need to solve a number of smaller MAXSAT problems.

Slicing the circuit. We can think of a circuit C as a sequence of subcircuits, or *slices*, $\langle C_0, \cdots, C_s \rangle$. Recall Fig. 4, which shows a circuit viewed as two slices. We will now demonstrate how to solve QMR by solving MAXSAT constraints for each slice in isolation.

We do this iteratively, starting with C_0 and going through the rest of the slices. First, for C_0, we simply generate a MAXSAT instance $(Hard_0, Soft_0)$ and solve it as described in the previous section. This results in a model I_0. Then for every slice C_i, where $i > 0$, we run the following procedure:

1) Generate MAXSAT constraints $(Hard_i, Soft_i)$ for C_i
2) For every variable $\mathsf{map}(q, p, |C_{i-1}|)$ set to true in I_{i-1}, add $\mathsf{map}(q, p, 1)$ to $Hard_i$.
3) Solve $(Hard_i, Soft_i)$, generating a model I_i

The interesting step here is step 2, which connects the final mapping from slice $i-1$ with the initial mapping from slice i. Specifically, we add the final mapping from slice $i-1$ as hard constraints on the initial mapping for slice i.

Example 9: Consider the circuit and connectivity graph in Fig. 6. If we solve QMR using the MAXSAT encoding, a possible optimal initial map is the one that maps q_i to p_i, as shown on the right, which requires no SWAPs to be inserted.

Suppose, however, that we slice the circuit into two slices, as highlighted. Solving the first slice might, for example, result in the map shown on the bottom right, with no swaps. This is optimal for the slice, but sub-optimal overall, since now we need to insert a SWAP between the two gates. Specifically, we will need to swap p_0 and p_1 (or p_2).

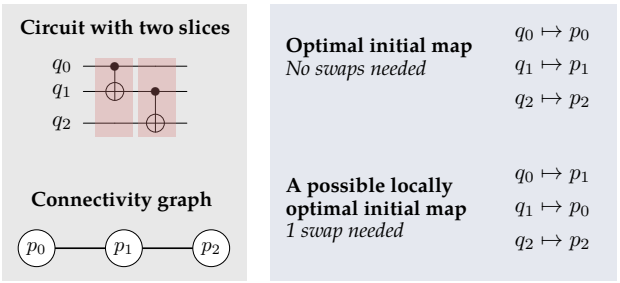


Fig. 6: Example demonstrating local relaxation

Backtracking. If our constraints allow a number of swaps n less than the diameter of the connectivity graph, then we could generate unsatisfiable formulas for some slices. In such cases, we backtrack to the previous slice and ask the MAXSAT solver to generate a different final mapping. Backtracking from a mapping involves adding the negation of its corresponding satisfying assignment (previously returned by the MAXSAT solver) as an additional hard constraint to ensure that the MAXSAT solver does not return the same mapping. After this additional constraint is added, backtracking is performed by re-invoking the solver.

Example 10: Consider again the map on the bottom right in Fig. 6. Let us suppose that we want to backtrack and find a different mapping from this one for the first slice. In order to guarantee that we do not return the same mapping again when re-invoking the MAXSAT solver, we add the following hard constraint:

$$\neg\big(\mathsf{map}(q_0,p_1,2) \wedge \mathsf{map}(q_1,p_0,2) \wedge \mathsf{map}(q_2,p_2,2)\big)$$

This constraint is exactly the negation of the encoding of the mapping that we wish to exclude.

VI. Exploiting Cyclic Circuits

For some quantum algorithms, the quantum circuits have a repeated structure, applying the same subcircuit multiple times. We call such circuits *cyclic* circuits.

The canonical algorithm that results in a cyclic circuit is the *quantum approximate optimization algorithm* (QAOA). QAOA is a general procedure for obtaining approximate solutions to NP-hard combinatorial problems such as determining a maximum cut in a graph. This is a promising near-term application since it solves problems of general practical interest and can be performed in the presence of noise without error-correction. The general structure of a QAOA circuit is shown in Fig. 7. Notice how the same subcircuit $C_{\gamma,\beta}$ repeats.[1]

A relaxation for cyclic circuits. For cyclic circuits, instead of solving a MAXSAT encoding for the entire circuit, we can relax the problem and only consider the repeating subcircuit. After finding an optimal solution for the subcircuit, we can *extend* the solution to the entire circuit. This results in a smaller

[1]Every cycle uses different parameters, γ,β, but the structure of the circuit remains the same, which is what matters for QMR. Also, the initial set of one-qubit gates (H) is irrelevant for QMR.

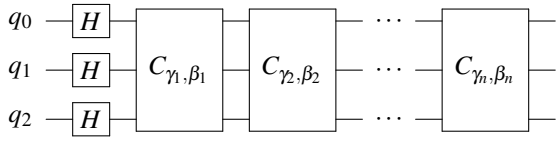


Fig. 7: The cyclic structure of a QAOA circuit

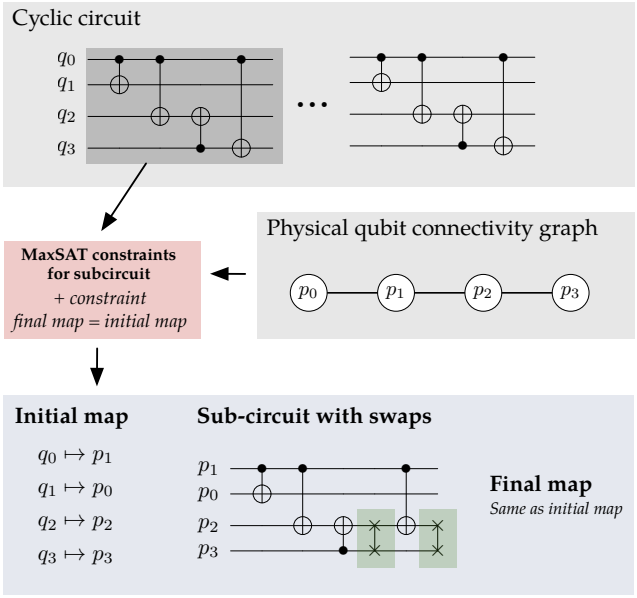


Fig. 8: Illustration of our cyclic circuit approach

MAXSAT problem that can generally be completed faster than the entire circuit, at the expense of a loss in optimality.

Suppose we have a circuit $\langle C,\ldots,C\rangle$, where the same subcircuit C is repeated a number of times, and a connectivity graph G. We follow the following simple recipe for solving the QMR problem; the key idea (step 2 below) is to ensure that the final map, $M_{|C|}$, is the same as the initial map, M_1, enabling us to *stitch* together two or more copies of C:

1) Let $(Hard, Soft)$ be the MAXSAT constraints for (C,G).
2) For every pair of logical and physical qubits, q and p, add the following constraint to $Hard$:

$$\mathsf{map}(q,p,1) \leftrightarrow \mathsf{map}(q,p,|C|)$$

3) Solve $(Hard, Soft)$, generating a model I.

The resulting circuit, with the initial map and swaps from I, can now be repeated an arbitrary number of times. This is because the map of logical to physical qubits is the same at the beginning and at the end.

Example 11: Fig. 8 revisits our running example, but with the same circuit iterated a number of times. Our cyclic-circuit approach in this case solves the same constraints as for Fig. 3, except that we add the hard constraint that the initial and final maps are the same. This results in the final circuit with two SWAPs, where the final SWAP is inserted to reset the mapping to its initial state (i.e., to swap back p_2 and p_3). Now this

subcircuit can be iterated any number of times on this physical connectivity graph.

This cyclic relaxation can be profitably combined with our local relaxation. Specifically, the MAXSAT constraints generated in step 1 for subcircuit C may be those that result from slicing it as described in Section V. Using such constraints, as we do in our evaluation (Section VII), allows our approach to handle larger subcircuits C.

VII. Implementation and Evaluation

Implementation. We implemented our approach in a tool we call SATMAP.[2] The value of n, the number of swaps allowed before each two-qubit gate, is set to 1. We experimentally determined $n = 1$ is sufficient for near-optimal solutions. SATMAP generates MAXSAT constraints and calls the MAXSAT solver Open-WBO-Inc-MCS [14] with default parameters. This solver provides the best known solution if it is interrupted before an optimal solution is found.

We convert all MAXSAT solutions to circuits. To ensure correctness of our QMR solutions, we implemented an independent verifier. The verifier traverses a circuit, evaluating its effects on an initial map and checking that all two-qubit gates act on connected qubits.

Throughout this section, whenever we say SATMAP, we imply that the locally optimal relaxation (Section V) is performed. The cyclic relaxation is turned off by default and evaluated later on cyclic circuits. The *slice size* refers to the number of two-qubit gates to include in each slice in the local relaxation. We always run SATMAP at four slice sizes, 10, 25, 50, and 100, and report the solution with the best cost. Solution costs are in terms of CNOT gates added (SWAP decomposes to 3 CNOTs).

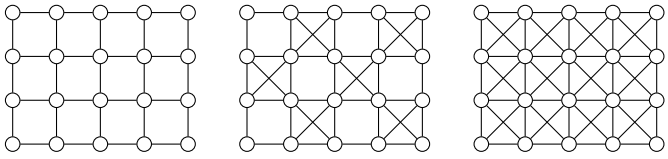
Benchmarks. For evaluation, we used the set of benchmarks collected in [10].[3] These 160 circuits were derived from the RevLib suite [15] and programs written in the Quipper [16] and ScaffoldCC [17] quantum programming languages. They cover a wide spectrum of circuit size, ranging in number of qubits from 3 to 16, and in two-qubit gates from 5 to over 200,000. The median number of two-qubit gates among these benchmarks is 123.

Except in the evaluation of Q4, the connectivity graph used is the IBM Q20 Tokyo architecture with 20 qubits, depicted in Fig. 9b. This connectivity graph was chosen as the largest typically used for evaluation in related work [3], [18], [19]. Benchmarks were evaluated on a cluster of Intel® Xeon® and AMD Opteron™ CPUs clocked an average of 2.5GHz.

Research questions. We designed a set of experiments to answer the following research questions:

- Q1 How does SATMAP compare to constraint-based techniques?
- Q2 How does SATMAP compare to heuristic approaches?
- Q3 What is the impact of local relaxation and cyclic circuit relaxation?

[2] https://github.com/qqq-wisc/satmap
[3] https://github.com/cda-tum/qmap/tree/main/examples



(a) Tokyo− **(b) IBM Q20 Tokyo** **(c) Tokyo+**

Fig. 9: Variations of the IBM Q20 Tokyo graph

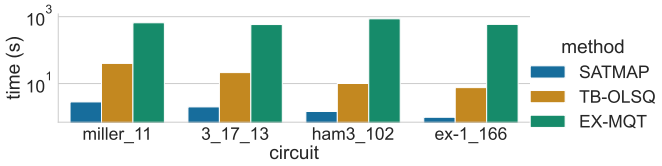


Fig. 10: Log-scale runtime comparison of EX-MQT, TB-OLSQ, and SATMAP on the set of benchmarks EX-MQT solved

- Q4 How does architecture variation impact performance?
- Q5 What is the scalability vs. optimality tradeoff?
- Q6 Can we use SATMAP with noise models?

Q1: Constraint-based approaches

Experimental setup. To address Q1, we compared SATMAP to the SMT-based tools EX-MQT [6] and TB-OLSQ [5]. The latter tool takes the relative execution time of each gate as input and can optimize several different objective functions. We set the execution time of each gate to 1 and the objective function to SWAP minimization to match our definition of QMR. For each of the benchmarks, SATMAP was allotted 30 minutes of compilation time and the other tools were each allotted 1 hour of compilation time to be as fair as possible and account for any potential hidden overheads. Each tool was allotted 5GB of RAM for each of the benchmarks.

Results. None of the tools were able to provide a solution to the QMR problem for all of the benchmarks within the time and memory restrictions. However, as Table I indicates, SATMAP handles a significantly higher proportion of the complete set, solving 109/160 (68%) of the benchmarks as compared to 4/160 (2.5%) solved by EX-MQT and 38/160 (24%) solved by TB-OLSQ. The additional benchmarks solved by SATMAP include circuits with up to 598 two-qubit gates versus a maximum of 23 for EX-MQT and 90 for TB-OLSQ.

Since these tools are all designed to provide optimal solu-

Tool	# Solved (out of 160)	Largest circuit solved
EX-MQT	4	23
TB-OLSQ	38	90
SATMAP	**109**	**598**

TABLE I: Comparison against constraint-based tools. Size of largest circuit solved is the number of two-qubit gates.

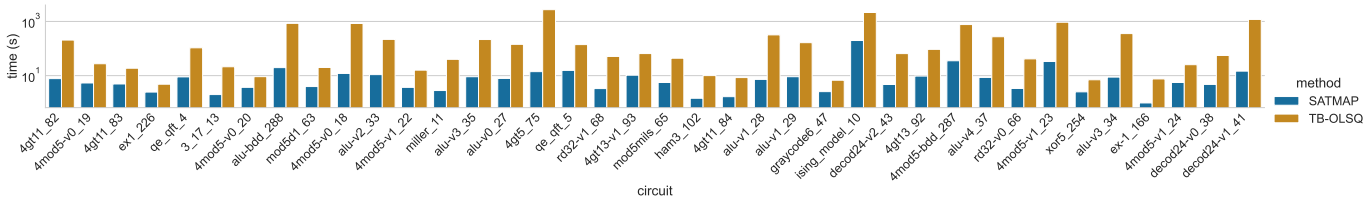


Fig. 11: Log-scale runtime comparison of TB-OLSQ and SATMAP on the complete set of benchmarks TB-OLSQ solved

tions, there is no variation in quality of the solutions.[4] Without sacrificing optimality, SATMAP significantly outperformed the two other tools in terms of runtime. The mean improvement over EX-MQT was about 400x and the mean improvement over TB-OLSQ was about 20x, as shown in Fig. 10 and Fig. 11.

Summary: SATMAP is significantly more scalable than EX-MQT and TB-OLSQ. It finds solutions 400x and 20x faster (respectively) and can be applied to much larger circuits, up to 598 two-qubit gates.

Q2: Heuristic approaches

Experimental setup. To address Q2, we compared SATMAP to the heuristic tools MQTH [10], SABRE [18], and TKET [3]. MQTH applies A* search to determine the optimal next mapping given the current one. SABRE and TKET both use heuristic "scores" to choose SWAPs to apply to the qubits relevant to a particular topological layer or timestep of a circuit, with SABRE applying a reversal procedure to determine a good initial mapping. These three were chosen to represent the state-of-the-art in terms of heuristic tools based on their widespread use in practice. Each is relied upon as a part of industrial quantum compilation toolkits. Since TKET and SABRE involve some element of nondeterminism, we took the mean cost of 20 runs of the heuristic tools. As in the evaluation of Q1, SATMAP was allotted 30 minutes of compilation time and 5GB of RAM for each of the benchmarks. The heuristic tools are less resource intensive, so they solved all of the instances well within those runtime and memory bounds.

Results. For the 109 benchmarks solved within the timeout by SATMAP, the resulting solutions were generally better than the heuristic tools. Fig. 12 presents the *cost ratio* on each benchmark: the total number of gates added by the heuristic tools divided by the total number of gates added by SATMAP. For each heuristic tool, there are rare instances (fewer than 10 benchmarks) when the heuristic outperforms SATMAP due to application of the local relaxation or early termination of the MAXSAT solver, resulting in ratios less than 1. On average, SATMAP adds 5.2x, 7.0x, and 3.6x fewer gates than MQTH, SABRE, and TKET, respectively. For all heuristic tools, there was at least one instance where SATMAP produced a solution with over 15x fewer gates. For about 14% of benchmarks, SATMAP did not add any gates, compared to 0%, 3%, and 10% for MQTH, SABRE, and TKET, respectively. Benchmarks where SATMAP added no gates and a heuristic tool added some gates are represented by the orange points at the top of the plot. Benchmarks where neither tool added gates have a cost ratio of 1.

Summary: when SATMAP terminates, it gives much higher quality solutions than heuristic tools overall. It almost always reduces the total cost—up to 6.97x on avg.

Q3: Impact of Relaxations

Local relaxation. We conducted experiments to determine the effect of the locally optimal relaxation (Section V) on performance in terms of execution time and cost. We tested the slice sizes, 10, 25, 50, and 100, against NL-SATMAP, which is SATMAP with local relaxation disabled.

Small slice sizes produce easier MAXSAT problems that can each individually be solved faster. However, restricting the "view" of the solver can lead to increased overall solve time due to repeated backtracking. We can observe this tradeoff by comparing the number of instances for which a solution was found within the 30 minute timeout (shown in Table II).

The situation is similar in terms of solution quality. For small slice sizes, local optima can diverge significantly from global optima. However, for moderate slice sizes, this effect is less pronounced. Fig. 13 presents the cost ratio of local relaxation levels: the total number of gates added by the tool with local relaxation divided by the total number of gates added by NL-SATMAP. For a slice size of 10, NL-SATMAP consistently produces better solutions, with an average cost ratio of 2.69. For larger slice sizes, the benchmarks where NL-SATMAP discovers better solutions are outnumbered by the benchmarks where the slow rate of convergence in a large solution space leads to worse solutions due to early termination. For example, with a slice size of 25, NL-SATMAP produces a better solution in 5 out of 70 cases, but a worse solution in 13 out of 70. This results in a mean ratio of less than 1 (0.92).

Summary: the local relaxation is a significant contributor to the performance of SATMAP. Appropriate application of local relaxation enables the use of constraint-based tools on large benchmarks, with little loss in solution quality for smaller ones.

Cyclic relaxation. To evaluate the cyclic relaxation, we programmatically generated a standard QAOA circuit for solving the maximum cut problem on 3-regular graphs, parameterized

[4] TB-OLSQ formulates QMR in terms of time coordinates, which treats a broader class of circuits as equivalent, allowing solutions not considered by the other tools. Additionally, a minor relaxation in TB-OLSQ produces suboptimal solutions in rare cases. The difference in cost due to these considerations is less than one SWAP in all cases.

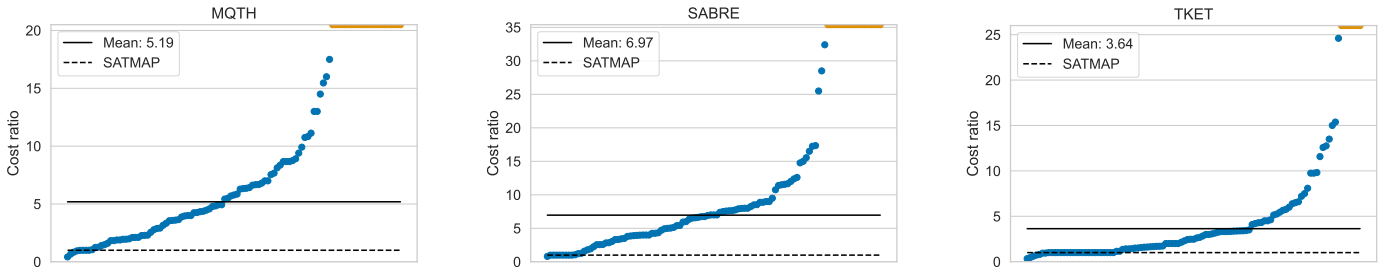


Fig. 12: The cost of the solution produced by each heuristic tool divided by the cost of the solution produced by SATMAP. Points at the top of the plot represent benchmarks where SATMAP added zero gates and the heuristic tool added a positive number, resulting in an undefined ratio. They are not included in the listed mean ratio.

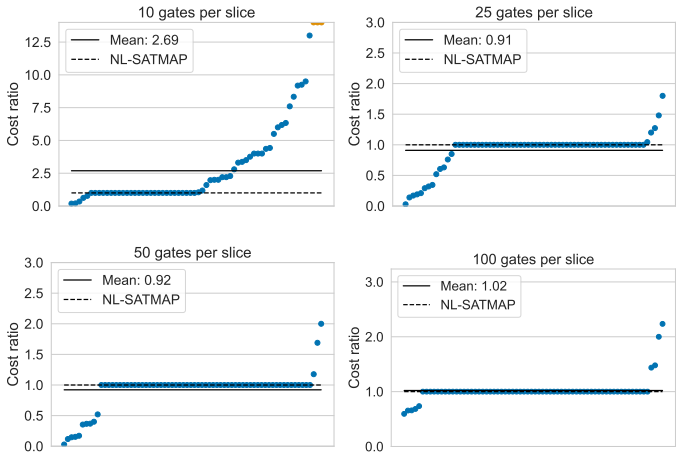


Fig. 13: The cost of the solution produced by the different levels of local relaxation divided by the cost of the solution produced by NL-SATMAP

Slice size	# Solved (out of 160)	Largest circuit solved
10	87	427
25	**103**	**598**
50	92	**598**
100	71	128
NL-SATMAP	70	128

TABLE II: Comparison of different levels of local relaxation in terms of instances solved

by the number of qubits and the number of cycles (repetitions of the subcircuit $C_{\gamma,\beta}$). We use CYC-SATMAP to denote SATMAP with the cyclic relaxation enabled.

We tested SATMAP, CYC-SATMAP, and TKET on QAOA circuits with 6, 8, 10, 12 and 16 qubits, each with two and four cycles. The results are presented in Table IV. Missing entries correspond to timeouts.

CYC-SATMAP solves all of the instances within the timeout, while SATMAP cannot come up with a solution for circuits with 10 qubits or 16 qubits. When it comes to cost, for 6 and 8 qubits, CYC-SATMAP outperforms SATMAP, whereas for 10 and

Tool	Main Set (total 160)		QAOA (total 10)	
	# Solved	Largest solved	# Solved	Largest solved
TB-OLSQ	38	90	0	–
NL-SATMAP	70	128	5	36
SATMAP	109	598	7	72
CYC-SATMAP	–	–	10	96

TABLE III: Comparison between TB-OLSQ, NL-SATMAP, SATMAP, and CYC-SATMAP

12 qubits, the opposite is true.[5] Additionally, except with 12 qubits, CYC-SATMAP determines a solution much more quickly than SATMAP. Neither SATMAP nor CYC-SATMAP has a clear advantage in terms of solution quality when both produce some solution. In some cases, CYC-SATMAP enables us to find a better solution than the best heuristic tool, TKET. For instance, with 16 qubits, CYC-SATMAP solutions are $> 3x$ better than TKET.

Summary: the cyclic relaxation improves performance of our approach on cyclic circuits in three respects: (1) it renders larger circuits tractable (such as 16-qubit QAOA), (2) it produces a solution faster, and (3) it produces better solutions within a fixed timeout for some circuits.

Breakdown of effects. Table III summarizes the effects of our encoding and relaxations, starting with TB-OLSQ as the baseline. Without relaxations, NL-SATMAP can solve 70 benchmarks, while TB-OLSQ only 38. With local relaxation, SATMAP can solve 109 benchmarks, with a largest circuit of size 598, compared to TB-OLSQ's 90. We also see the same behavior in QAOA benchmarks, with the local relaxation (SATMAP) and cyclic relaxations (CYC-SATMAP) allowing us to solve progressively more benchmarks. TB-OLSQ is unable to solve any of our QAOA benchmarks within the allotted time.

Q4: Impact of Architecture

Experimental setup. Finally, we investigated the effectiveness of SATMAP as compared to the best heuristic tool, TKET, when varying the properties of the connectivity graph. We constructed

[5]Note that SAT solvers do not provide a monotonicity guarantee on performance with respect to increasing circuit size—e.g., we can solve a 12-qubit circuit with SATMAP but not a 10-qubit circuit. This is due to the search strategy employed by the underlying solver.

# Qubits	Cycles	CYC-SATMAP		SATMAP		TKET	
		Cost	Time	Cost	Time	Cost	Time
6	2	**12**	130	**12**	1800	**12**	< 0.1
	4	**24**	130	60	203	42	< 0.1
8	2	**12**	361	18	954	21	< 0.1
	4	**24**	361	63	372	30	< 0.1
10	2	84	253	54	1155	**33**	0.14
	4	168	253	–	–	**102**	0.24
12	2	84	1800	**21**	261	48	0.33
	4	168	1800	105	1800	**87**	0.32
16	2	**24**	288	–	–	78	0.30
	4	**48**	288	–	–	147	0.41

TABLE IV: Quality of solutions and runtime (s) of CYC-SATMAP, SATMAP, and TKET on QAOA circuits

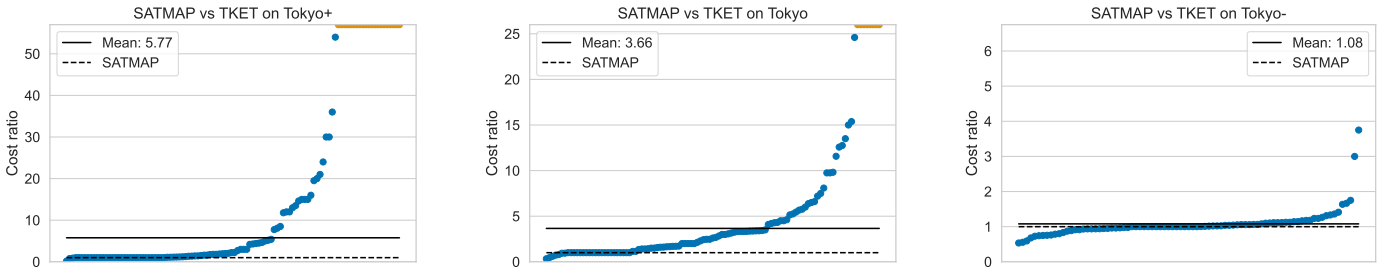


Fig. 14: The cost of the solution produced by TKET divided by the cost of the solution produced by SATMAP on three different connectivity graphs: Tokyo+, Tokyo, and Tokyo−

two modified versions of the IBM Tokyo architecture: (1) increasing sparsity of the graph by removing diagonal edges (Tokyo−, Fig. 9a), and (2) increasing connectivity by adding diagonal edges (Tokyo+, Fig. 9c). The average degree of a vertex in Tokyo is exactly halfway between Tokyo+ and Tokyo−. We applied the same procedure as in the evaluation of Q2 with the use of these different architectures as the only difference.

Results. In a similar manner to Q2, Fig. 14 shows the cost ratio on each benchmark for the three architectures. On Tokyo−, heuristic tools and SATMAP produce very similar solutions. The difference between the cost of the solution produced by TKET and SATMAP was less than 10 gates for 61 of the 85 benchmarks solved by SATMAP, with a mean cost ratio near 1. Results on Tokyo+ are more in line with those on Tokyo, but with more variance across the benchmark set. Again comparing TKET to SATMAP, the standard deviation in cost ratio on Tokyo+ is 9.09 as opposed to 3.92 on Tokyo (excluding infinite ratios). These results suggest the existence of two effects. First, heuristic solutions are well-suited to finding near optimal solutions on sparse connectivity graphs. Second, the success of SATMAP on Tokyo as compared to Tokyo− and Tokyo+ may indicate constraint-based tools are better suited to non-uniform architectures where the connectivity varies across qubits. We observe the same behavior on SABRE and MQTH, so we focus on TKET here since it's the best-performing tool.

Summary: heuristic-based tools are not robust to variations in the connectivity graph, tending to produce better results on sparse graphs (Tokyo−) than highly connected ones (Tokyo+).

Q5: Scalability and Optimality

Time Limits. First, we study the impact of the time bound on the number of benchmarks solved and the cost of the solution. We consider the following time limits (in seconds): 100, 300, 600, 1800, 3600, 5400, and 7200. We compare each time limit against the original 1800 seconds by computing the cost ratio like in Q2. All other configurations are the same including the architecture, Tokyo. As expected, the cost ratio decreases exponentially as the time allotted increases meaning solution quality improves with more time. Additionally, the number of benchmarks solved and size of the largest circuit solved both increase given more time. The change in the number of benchmarks solved across time is less dramatic, increasing from 103 to 111.

Cost vs Circuit Size. Second, we analyze the optimality of the solution compared to circuit size. Since we do not have ground truth optimal cost, we use the cost ratio data from Q2 comparing SATMAP to the best performing heuristic tool, TKET. We observe a downward trend in cost ratio as circuit size increases, suggesting a loss in optimality as circuit size

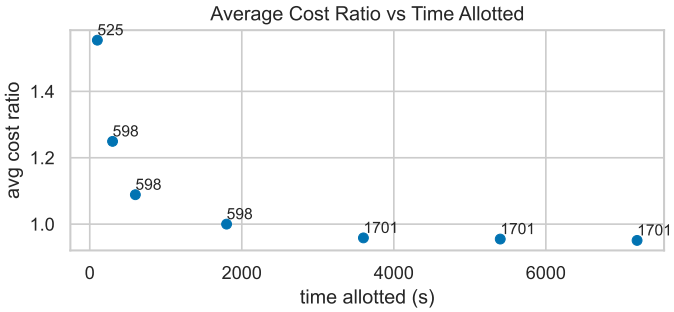


Fig. 15: Comparison of average cost ratio across different time bounds with a baseline of 1800 seconds. Each point is labeled with the size of the largest circuit solved.

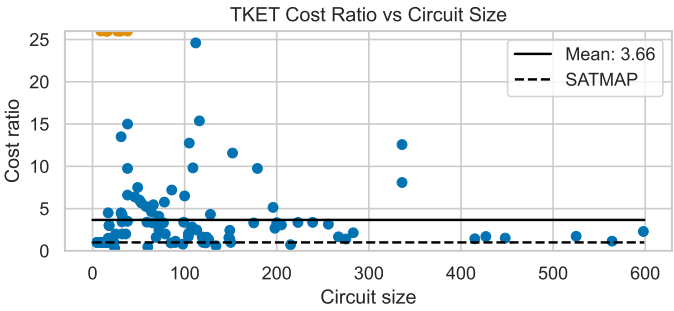


Fig. 16: Solution quality compared to TKET across different circuit sizes.

increases. This is expected as the local relaxation creates more slices for larger circuits.

Q6: Noise Models

To demonstrate the versatility of our approach, we also use a *weighted* MAXSAT encoding to incorporate noise models, with the aim of maximizing the *fidelity* of the output circuit.

Weighted MAXSAT is a generalization of MAXSAT where soft constraints are assigned a positive weight, and the objective is to maximize the sum of the weights of the satisfied soft clauses. The MAXSAT problem as defined previously is equivalent to weighted MAXSAT where all weights are 1.

Example 12: Consider the weighted MAXSAT problem with one hard constraint and two soft constraints,

$$Hard = \{a \vee b\}$$
$$Soft = \{(\neg a, weight : 5), (\neg b, weight : 1)\}$$

The solution is $I = [a \mapsto \mathsf{false}, b \mapsto \mathsf{true}]$, with a weight of 5.

In the noise-aware version of SATMAP, we use soft-clause weights to encode the fidelity of operations, e.g, a variable $\mathsf{swap}(p_1, p_2, k)$ is assigned a weight (probability between 0 and 1) corresponding to the fidelity of performing a SWAP on the edge (p_1, p_2). The result is an optimization objective equivalent to TB-OLSQ's. We used the error rates supplied by the "FakeTokyo" backend from the IBM Qiskit development kit.

Fidelity maximization is a more complex objective than SWAP minimization, so both tools, SATMAP and TB-OLSQ, solved fewer benchmarks from the set of 160 within the same timeout from Q1. However, we observe an even bigger gap between the two tools, with SATMAP solving ∼4x more benchmarks: The fidelity maximization version of TB-OLSQ was able to solve 23 benchmarks, whereas SATMAP was able to solve 89. For the 23 benchmarks solved by TB-OLSQ, the fidelity achieved by by SATMAP was the same except for 5 examples, which incur a small fidelity reduction of 0.004 to 0.09 due to relaxation.

VIII. Related Work

Constraint-based Approaches. Many prior works utilize constraint solvers for mapping and routing logical qubits. Several tools leverage ILP solvers to minimize the number of SWAPs and circuit depth [20], [21], [22]. At the same time, several others translate QMR to Boolean constraints and use SMT solvers [6], [23]. Besides restricted connectivity, mapping logical qubits onto physical qubits can be challenging due to non-uniform gate latency and error rates. To that end, Tan et al. leverage SMT solvers to minimize the total circuit runtime [5] whereas Murali et al. use SMT solvers to maximize success probability by accounting for variability in gate errors [7]. Furthermore, some tools cast QMR as a path planning problem and leverage temporal planners [24], [25]. However, most constraint-based solvers face severe scalability issues due to the exponential search space.

The closest work to ours is TB-OLSQ [5] which uses a *satisfiability modulo theories* (SMT) encoding that is more efficient than earlier work [6]. Our encoding is different in a number of ways: (1) We restrict ourselves to fully Boolean encoding, allowing us to sidestep the complexity of SMT vs SAT solving. (2) We model SWAPs via Boolean variables in a view that mimics sketch-based program synthesis. (3) We introduce novel relaxations that increase scalability while maintaining almost optimality.

Heuristic-based Approaches. Due to the limited scalability of constraint-based tools, most industry compilers and open-source quantum compiler projects use heuristic methods for performing QMR. For example, IBM Qiskit uses SABRE, a bidirectional local search algorithm that slices the circuit into subcircuits and finds locally optimal mappings, similar to our locally optimal relaxation [18]. In contrast, the MQT compiler uses a slow but exhaustive A* search [10]; the approach is made feasible by only applying A* between gates in consecutive topological layers of the circuit. Furthermore, recent work combines A* with novel search space pruning techniques to enable time optimal QMR solutions [26]. Tools like Enfield use subgraph isomorphism and token swapping [19], whereas others use hierarchical product algorithms for modular architecture [27] and SWAP networks [28]. Majority of the heuristic methods try to minimize the distance between logical qubits and use greedy but efficient local search to reduce the number of gates and circuit depth. For example, TKET heuristic performs a greedy search to find an initial qubit mapping that results in the least number of SWAPs and inserts routing gates by iteratively

permuting the logical to physical mapping [3], [29]. Similarly, earlier work uses local permutations of physical to logical qubit map and sub-graph isomorphism to solve the QMR problem [30]. A large body of work focuses on restricted qubit architectures with 1D, and 2D nearest neighbor connectivity [31], [32], [33], [34]. Whereas recent works focus on IBM machines and develop greedy search strategies for IBM architectures [35], [36]. While others develop application-specific [37], [38] and noise-aware strategies that use hardware-level characteristics to improve fidelity [39], [40], [41], [42], [43], [44], [45].

IX. Discussion

Importance of optimality. Our results demonstrate a big gap in terms of added SWAPs between heuristic-based QMR algorithms and our constraint-based technique. For near-term, noisy quantum computers, reducing the number of SWAPs is critical for successfully executing quantum algorithms. Therefore, our results indicate that, going forward, (1) we need to improve existing heuristic algorithms to bring them closer to optimal or (2) improve the scalability of constraint-based techniques.

Scaling the MAXSAT approach. Our MAXSAT approach produces significant improvements over existing optimal approaches. However, SAT solving is an NP-complete problem and scalability remains an issue. We see two avenues for scaling our MAXSAT approach to stay ahead of the growth in the number of qubits: First, we can employ parallel SAT-solving strategies. All of our experiments used a single-threaded solver. Today, there are SAT solvers that can run on cloud infrastructure with impressive improvements.[6] Second, at the algorithmic level, we can combine our MAXSAT approach with heuristic approaches. For instance, we can only solve the mapping constraints (optimally) and leave the routing process for a heuristic approach or an approximation algorithm [23].

Future architectures. Quantum computing is a field in flux, and there is no clear indication of how future connectivity graphs will look like, as it depends on the underlying physical substrate used and engineering advances. The ideal, of course, is to build a device with as much connectivity and as little cross-talk as possible. Our results demonstrate that for higher connectivity, the performance of heuristic approaches diverges from the optimal. So it is prudent to robustify heuristic approaches to changes in the connectivity graph to accommodate the rapid developments NISQ computing hardware.

Acknowledgments

We thank the anonymous reviewers for their thoughtful feedback and Rui Huang for providing scripts to generate QAOA circuits. This work is supported by NSF grants #1652140 and #2212232 and awards from Meta and Amazon. This research is also partially supported by the Vice Chancellor Office for Research and Graduate Education at the University of Wisconsin–Madison with funding from the Wisconsin Alumni Research Foundation.

[6] https://satcompetition.github.io/2021/

References

[1] M. A. Nielsen and I. Chuang, "Quantum computation and quantum information," 2002.

[2] J. Preskill, "Quantum computing in the nisq era and beyond," *Quantum*, vol. 2, p. 79, 2018.

[3] A. Cowtan, S. Dilkes, R. Duncan, A. Krajenbrink, W. Simmons, and S. Sivarajah, "On the qubit routing problem," *arXiv preprint arXiv:1902.08091*, 2019.

[4] B. Tan and J. Cong, "Optimality study of existing quantum computing layout synthesis tools," *IEEE Transactions on Computers*, vol. 70, no. 9, pp. 1363–1373, 2020.

[5] ——, "Optimal layout synthesis for quantum computing," in *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE, 2020, pp. 1–9.

[6] R. Wille, L. Burgholzer, and A. Zulehner, "Mapping quantum circuits to ibm qx architectures using the minimal number of swap and h operations," in *2019 56th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2019, pp. 1–6.

[7] P. Murali, J. M. Baker, A. J. Abhari, F. T. Chong, and M. Martonosi, "Noise-adaptive compiler mappings for noisy intermediate-scale quantum computers," *arXiv preprint arXiv:1901.11054*, 2019.

[8] A. Biere, M. Heule, and H. van Maaren, *Handbook of satisfiability*. IOS press, 2009, vol. 185.

[9] J. Backes, P. Bolignano, B. Cook, C. Dodge, A. Gacek, K. Luckow, N. Rungta, O. Tkachuk, and C. Varming, "Semantic-based automated reasoning for aws access policies using smt," in *2018 Formal Methods in Computer Aided Design (FMCAD)*. IEEE, 2018, pp. 1–9.

[10] A. Zulehner, A. Paler, and R. Wille, "An efficient methodology for mapping quantum circuits to the ibm qx architectures," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 7, pp. 1226–1236, 2018.

[11] A. Solar-Lezama, *Program synthesis by sketching*. University of California, Berkeley, 2008.

[12] E. Farhi, J. Goldstone, and S. Gutmann, "A quantum approximate optimization algorithm," *arXiv preprint:1411.4028*, 2014.

[13] I. P. Gent and P. Nightingale, "A new encoding of alldifferent into sat," in *International Workshop on Modelling and Reformulating Constraint Satisfaction*, 2004, pp. 95–110.

[14] S. Joshi, P. Kumar, V. Manquinho, R. Martins, A. Nadel, and S. Rao, "Open-WBO-Inc in MaxSAT Evaluation 2018," in *MaxSAT Evaluation 2018: Solver and Benchmark Descriptions*, vol. B-2018-2. Department of Computer Science, University of Helsinki, 2018, pp. 16–17.

[15] R. Wille, D. Große, L. Teuber, G. W. Dueck, and R. Drechsler, "Revlib: An online resource for reversible functions and reversible circuits," in *38th International Symposium on Multiple Valued Logic (ismvl 2008)*, 2008, pp. 220–225.

[16] A. S. Green, P. L. Lumsdaine, N. J. Ross, P. Selinger, and B. Valiron, "Quipper: A scalable quantum programming language," *CoRR*, vol. abs/1304.3390, 2013. [Online]. Available: http://arxiv.org/abs/1304.3390

[17] A. JavadiAbhari, S. Patil, D. Kudrow, J. Heckey, A. Lvov, F. T. Chong, and M. Martonosi, "Scaffcc: A framework for compilation and analysis of quantum computing programs," in *Proceedings of the 11th ACM Conference on Computing Frontiers*, ser. CF '14. New York, NY, USA: Association for Computing Machinery, 2014. [Online]. Available: https://doi.org/10.1145/2597917.2597939

[18] G. Li, Y. Ding, and Y. 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*, 2019, pp. 1001–1014.

[19] M. Y. Siraichi, V. F. d. Santos, C. Collange, and F. M. Q. Pereira, "Qubit allocation as a combination of subgraph isomorphism and token swapping," *Proceedings of the ACM on Programming Languages*, vol. 3, no. OOPSLA, pp. 1–29, 2019.

[20] D. Bhattacharjee, A. A. Saki, M. Alam, A. Chattopadhyay, and S. Ghosh, "Muqut: Multi-constraint quantum circuit mapping on nisq computers," in *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2019, pp. 1–7.

[21] A. Chakrabarti, S. Sur-Kolay, and A. Chaudhury, "Linear nearest neighbor synthesis of reversible circuits by graph partitioning," *arXiv preprint arXiv:1112.0564*, 2011.

[22] D. Bhattacharjee and A. Chattopadhyay, "Depth-optimal quantum circuit placement for arbitrary topologies," *arXiv preprint arXiv:1703.08540*, 2017.

[23] M. Y. Siraichi, V. F. d. Santos, S. Collange, and F. M. Q. Pereira, "Qubit allocation," in *Proceedings of the 2018 International Symposium on Code Generation and Optimization*. ACM, 2018, pp. 113–125.

[24] D. Venturelli, M. Do, E. G. Rieffel, and J. Frank, "Temporal planning for compilation of quantum approximate optimization circuits." in *IJCAI*, 2017, pp. 4440–4446.

[25] D. Venturelli, M. Do, E. Rieffel, and J. Frank, "Compiling quantum circuits to realistic hardware architectures using temporal planners," *Quantum Science and Technology*, vol. 3, no. 2, p. 025004, 2018.

[26] C. Zhang, A. B. Hayes, L. Qiu, Y. Jin, Y. Chen, and E. Z. Zhang, "Time-optimal qubit mapping," in *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2021, pp. 360–374.

[27] A. M. Childs, E. Schoute, and C. M. Unsal, "Circuit transformations for quantum architectures," *arXiv preprint arXiv:1902.09102*, 2019.

[28] B. O'Gorman, W. J. Huggins, E. G. Rieffel, and K. B. Whaley, "Generalized swap networks for near-term quantum computing," *arXiv preprint arXiv:1905.05118*, 2019.

[29] S. Sivarajah, S. Dilkes, A. Cowtan, W. Simmons, A. Edgington, and R. Duncan, "t|ket⟩: a retargetable compiler for nisq devices," *Quantum Science and Technology*, vol. 6, no. 1, p. 014003, 2020.

[30] D. Maslov, S. M. Falconer, and M. Mosca, "Quantum circuit placement: optimizing qubit-to-qubit interactions through mapping quantum circuits into a physical experiment," in *Proceedings of the 44th annual Design Automation Conference*. ACM, 2007, pp. 962–965.

[31] Y. Hirata, M. Nakanishi, S. Yamashita, and Y. Nakashima, "An efficient conversion of quantum circuits to a linear nearest neighbor architecture," *Quantum Information & Computation*, vol. 11, no. 1, pp. 142–166, 2011.

[32] M. Saeedi, R. Wille, and R. Drechsler, "Synthesis of quantum circuits for linear nearest neighbor architectures," *Quantum Information Processing*, vol. 10, no. 3, pp. 355–377, 2011.

[33] A. Shafaei, M. Saeedi, and M. Pedram, "Optimization of quantum circuits for interaction distance in linear nearest neighbor architectures," in *2013 50th ACM/EDAC/IEEE Design Automation Conference (DAC)*. IEEE, 2013, pp. 1–6.

[34] R. Wille, O. Keszocze, M. Walter, P. Rohrs, A. Chattopadhyay, and R. 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)*. IEEE, 2016, pp. 292–297.

[35] A. Zulehner and R. Wille, "Compiling su (4) quantum circuits to ibm qx architectures," in *Proceedings of the 24th Asia and South Pacific Design Automation Conference*, 2019, pp. 185–190.

[36] A. Kole, S. Hillmich, K. Datta, R. Wille, and I. Sengupta, "Improved mapping of quantum circuits to ibm qx architectures," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 10, pp. 2375–2383, 2019.

[37] G. G. Guerreschi and J. Park, "Two-step approach to scheduling quantum circuits," *Quantum Science and Technology*, vol. 3, no. 4, p. 045003, 2018.

[38] M. Alam, A. Ash-Saki, and S. Ghosh, "Circuit compilation methodologies for quantum approximate optimization algorithm," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2020, pp. 215–228.

[39] S. S. Tannu and M. 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*, 2019, pp. 987–999.

[40] P. Das, S. Tannu, S. Dangwal, and M. Qureshi, "Adapt: Mitigating idling errors in qubits via adaptive dynamical decoupling," in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, 2021, pp. 950–962.

[41] P. Das, S. Tannu, and M. Qureshi, "Jigsaw: Boosting fidelity of nisq programs via measurement subsetting," in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, 2021, pp. 937–949.

[42] P. Jurcevic, A. Javadi-Abhari, L. S. Bishop, I. Lauer, D. F. Bogorin, M. Brink, L. Capelluto, O. Günlük, T. Itoko, N. Kanazawa *et al.*, "Demonstration of quantum volume 64 on a superconducting quantum computing system," *Quantum Science and Technology*, vol. 6, no. 2, p. 025020, 2021.

[43] S. S. Tannu and M. Qureshi, "Ensemble of diverse mappings: Improving reliability of quantum computers by orchestrating dissimilar mistakes," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2019, pp. 253–265.

[44] T. Patel and D. Tiwari, "Veritas: accurately estimating the correct output on noisy intermediate-scale quantum computers," in *2020 SC20: International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*. IEEE Computer Society, 2020, pp. 188–203.

[45] S. Stein, N. Wiebe, Y. Ding, P. Bo, K. Kowalski, N. Baker, J. Ang, and A. Li, "Eqc: ensembled quantum computing for variational quantum algorithms," in *Proceedings of the 49th Annual International Symposium on Computer Architecture*, 2022, pp. 59–71.