Towards Optimal Topology Aware Quantum Circuit Synthesis

Marc G. Davis, Ethan Smith, Ana Tudor, Koushik Sen, Irfan Siddiqi
University of California Berkeley
{marc.davis,ethanhs,anamtudor,ksen,irfan_siddiqi}@berkeley.edu

Costin Iancu
Lawrence Berkeley National Laboratory
cciancu@lbl.gov

***Abstract*—We present an algorithm for compiling arbitrary unitaries into a sequence of gates native to a quantum processor. As CNOT gates are error-prone for the foreseeable Noisy-Intermediate-Scale Quantum devices era, our A* inspired algorithm minimizes their count while accounting for connectivity. We discuss the formulation of synthesis as a search problem as well as an algorithm to find solutions. For a workload of circuits with complexity appropriate for the NISQ era, we produce solutions well within the best upper bounds published in literature and match or exceed hand tuned implementations, as well as other existing synthesis alternatives. In particular, when comparing against state-of-the-art available synthesis packages we show $2.4\times$ average (up to $5.3\times$) reduction in CNOT count. We also show how to re-target the algorithm for a different chip topology and native gate set while obtaining similar quality results. We believe that tools like ours can facilitate algorithmic exploration and guide gate set discovery for quantum processor designers, as well as being useful for optimization in the quantum compilation tool-chain.**

I. Introduction

There is a high probability that quantum computing will deliver transformational scientific results within the next few decades. Right now, we are in an era of effervescence, where the first available [1]–[3] hardware implementations of quantum processors have opened the doors for exploration in quantum hardware, software, and algorithmic design. All three lines of inquiry have in common that the unitary matrix associated with the transformation (algorithm, gate, circuit etc.) is readily obtained, while deriving an efficient circuit to represent a desired transformation is generally difficult. Quantum circuit synthesis is an approach to derive a circuit that implements a given unitary and can thus facilitate advances in all these directions: hardware, software and algorithmic exploration.

Research into quantum circuit synthesis has a long [4]–[12] history. Synthesis can be a tool of great utility in the quantum development kit for the Noisy Intermediate-Scale Quantum (NISQ) Devices era, which is characterized by design space exploration at small qubit scale, together with a need for highly optimized implementations of circuits. To foster adoption, tools need to overcome some of the currently perceived shortcomings:

- Synthesis generates deep circuits
- Synthesis does not account for hardware topology
- Synthesis is slow

In this paper we describe a pragmatic synthesis algorithm designed to minimize the number of CNOT gates in the resulting circuit. As CNOT has low fidelity on existing hardware and is expected to be the limiting factor in the near future of NISQ devices, the CNOT count metric has been a frequent target for investigation [12]–[15]. We show how to generate short CNOT count circuits while accounting for chip topology by: 1) posing the problem of finding a circuit structure as a tree search and using a highly effective algorithm for shortest path problems; 2) combining the search with numerical optimization for instantiation of circuit parameters.

The algorithm is inspired by the A* [16] search strategy and works as follows. We start building the circuit in layers: at each step we add a fixed structure 2-qubit building block where *chip connectivity* allows it. The building block contains generic parameterized single qubit unitaries and native 2-qubit gates, in our case CNOT . We pass the parameterized circuit into an optimizer [17], which instantiates the parameters for the partial solution such that it minimizes a distance function. At each step of the search, a heuristic is used to select a circuit to expand. The algorithm stops when the current solution is indistinguishable from the target unitary. We now have a concrete circuit that can be implemented on hardware.

We target two superconducting qubit architectures: the QNL8QR-V5 chip developed by the UC Berkeley Quantum Nanoelectronics Laboratory [18], with eight superconducting qubits connected in a line topology and the IBM Q5 [19] chip with qubits connected in a "bowtie". Both chips have a similar native gate set composed of single qubit rotations and CNOT gates. For evaluation we use known algorithms and gates, e.g. QFT, HHL, Fredkin, Toffoli etc., with implementations obtained from other researchers [20].

Our approach offers several contributions that advance the state of the art in quantum circuit synthesis. First, the data dispels the concern that synthesis produces deep circuits. A* search combined with numerical optimization produces nearly optimal depth circuits. When comparing against circuits that were hand-optimized, our implementation matches or reduces the CNOT count. When comparing against state-of-the-art generic synthesis tools such as UniversalQ [21], our circuits are shorter, with $2.4\times$ average reduction in CNOT gates, and by as much as $5.3\times$. When comparing against state-of-the-art domain specific optimizers [22] we reduce circuit depth by as much as $5\times$.

Second, to our knowledge we provide the first practical demonstration of topology aware synthesis that does not significantly increase circuit length. Specializing the search strategy for a given topology results in circuits than do not need additional SWAP operations inserted at the mapping stage. Existing approaches assume all-to-all connectivity, and modifications to handle restricted topologies introduce large (e.g. $4\times$ [13]) proportionality constants. In our case, we observe only modest differences between circuits synthesized for all-to-all and circuits synthesized for the linear topology. When specializing for the linear topology, we observe increased CNOT count on only five circuits (half workload), with an average of 15% increase for the whole workload. Furthermore, the depth difference from topology customization cannot be recuperated by the rest of the optimization toolchain: the final depth of a circuit synthesized for all-to-all topology and then mapped for the linear topology by IBM QISKit, is longer than the depth of the circuit synthesized directly for the linear topology. When using circuit mapping to specialize for topology, we observe a 53% average increase in CNOT count, and up to 4×, compared to synthesizing directly for the linear topology.

We also show how our infrastructure can be easily retargeted to different native gate sets, qutrit [23] based circuits and to generate state preparation circuits. To our knowledge, this is the first demonstration of synthesis of multi-gate multi-qutrit based circuits. When comparing our state preparation results against IBM Qiskit, we observe circuits shorter by 36% and KL divergence smaller by 4.8% on average. The results indicate that synthesis can be a very useful tool in the stack of quantum circuit compilation tools.

The rest of this paper is structured as follows. In Section II we introduce the problem, its motivation and provide a short primer on quantum computing. In Section III we describe our algorithm and its implementation, while in Section IV-A we present results for the three usage scenarios. In Section V we discuss future uses of synthesis in the NISQ era, while in Section VI we describe the related work.

II. Background

In quantum computing, a qubit is the basic unit of quantum information. Physically, qubits are two-level quantum-mechanical systems, whose general quantum state is represented by a linear combination of two orthonormal basis states (basis vectors). The most common basis is the equivalent of the 0 and 1 values used for bits in classical information theory, respectively $|0\rangle = \begin{bmatrix}1\\0\end{bmatrix}$ and $|1\rangle = \begin{bmatrix}0\\1\end{bmatrix}$. The generic qubit state is a superposition of the basis states, i.e. $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$, with complex amplitudes α and β such that $|\alpha|^2 + |\beta|^2 = 1$.

A. Gate Sets in Quantum Computing

The prevalent model of quantum computation is the circuit model introduced by [24], where information carried by qubits (wires) is modified by quantum gates, which mathematically correspond to unitary operations. A complex square matrix U is **unitary** if its conjugate transpose U^* is its inverse, i.e. $UU^* = U^*U = I$.

In the circuit model, a single qubit gate is represented by a 2×2 unitary matrix U. The effect of the gate on the qubit state is obtained by multiplying the U matrix with the vector representing the quantum state $|\psi'\rangle = U|\psi\rangle$.

The most general form of the unitary for a single qubit gate is the "continuous" or "variational" gate representation.

$$U3(\theta,\phi,\lambda) = \begin{pmatrix} cos\frac{\theta}{2} & -e^{i\lambda}sin\frac{\theta}{2} \\ e^{i\phi}sin\frac{\theta}{2} & e^{i\lambda+i\phi}cos\frac{\theta}{2} \end{pmatrix} \tag{1}$$

In quantum computing theory, a set of quantum gates is *universal* if any computation (unitary transformation) can be approximated on any number of qubits to any precision when using only gates from the set. On the hardware side, quantum processors expose a set of native gates which constitute a universal set. Quantum processors built from superconducting qubits usually provide single qubit rotations (R_x, R_y, and R_z) and two qubit CNOT , CZ or SWAP gates.

A CNOT , or controlled NOT gate, flips the target qubit *iff* the control qubit is $|1\rangle$. CNOT has the following unitary:

$$CNOT = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \tag{2}$$

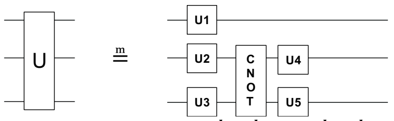
A circuit is described by an evolution in space (application on qubits) and time of gates. Figure 1 shows an example circuit that applies single qubit and CNOT gates on three qubits.

B. Background on Quantum Circuit Synthesis

A quantum transformation (algorithm, circuit) on n qubits is represented by a unitary matrix U of size $2^n \times 2^n$. The goal of circuit synthesis is to decompose U into a product of terms, where each individual term captures the application of a quantum gate on individual qubits. This is depicted in Figure 1. The quality of a synthesis algorithm is evaluated by the number of gates in the circuit it produces and by the distinguishability of the solution from the original unitary. We discuss in more detail related work in synthesis in Section VI and summarize in this section only the pertinent state-of-the-art results for NISQ devices.

Circuit length provides the optimality criteria for synthesis algorithms: shorter circuits are better. CNOT count is a direct indicator of overall circuit length, as the number of single qubit generic gates introduced in the circuit is proportional with a constant given by decomposition rules. Thus CNOT count or circuit length can be used interchangeably when discussing optimality criteria. As CNOT gates are problematic on NISQ devices, state-of-the-art approaches [13], [15] directly attempt to minimize their count.

There are two main types of synthesis approaches: unitary decomposition using linear algebra techniques, and empirical search based techniques. The state-of-the-art linear algebra techniques use Cosine-Sine Decomposition [13], [15] and provide upper bounds on circuit depth. We use the tightest published upper bounds for the evaluation of our approach, as well as direct comparisons with the UniversalQ [21] compiler, which implements these algorithms. Empirical approaches use search heuristics for decomposition. We discuss these in Section VI. Most existing algorithms are not often used in practice because they produce long circuits and lack topology



$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \otimes \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} = \begin{bmatrix} a_{11}\begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} & a_{12}\begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} \\ a_{21}\begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} & a_{22}\begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} \end{bmatrix}$$

Fig. 1. *Unitaries (above) and tensors products (below). The unitary U represents a $n = 3$ qubit transformation, where U is a $2^n \times 2^n$ (8×8) matrix. The unitary is implemented (equivalent or approximated) by the circuit on the right hand side. The single qubit unitaries are 2×2 matrices, while CNOT is a $2^2 \times 2^2$ matrix. The computation performed by the circuit is $(I2 \otimes U4 \otimes U5)(I2 \otimes CNOT)(U1 \otimes U2 \otimes U3)$, where I2 is the identity 2×2 matrix and $\otimes$ is the tensor product operator. The right hand side shows the tensor product of 2×2 matrices.*

awareness. The notable exception is the ubiquitous deployment of KAK [12] optimal 2-qubit unitary decomposition in commercial [25]–[27] compilers.

Table I presents the best known upper bounds on CNOT *count for synthesis algorithms.* Note that for three qubits the bound is 20 CNOT, while for four qubits it is 100 CNOT. Asymptotically, the tightest bound is introduced by [13] to a CNOT count of $0.16 * (4^m + 2 * 4^n)$. Because of the exponentiation, for current generation devices it is important to demonstrate quantitatively that we can attain shorter depth.

Taking chip qubit connectivity into account during synthesis affects circuit depth. Most algorithms implicitly assume full qubit connectivity, and therefore may place CNOT gates between qubits that are not physically connected. In these cases, a mapping algorithm is needed to introduce SWAP gates, with each SWAP gate implemented using three CNOT gates. Recent approaches try to provide bounds when specializing for topology by estimating the number of additional SWAPs. The algorithms presented by [15] increase the CNOT count by a factor of nine when restricting topology to a nearest-neighbor (linear topology) interaction, while [13] claim a factor of four.

n m	0	1	2	3	4
2	1	2	**3**	-	-
3	3	9	14	**20**	-
4	8	22	54	73	**100**

TABLE I
Upper bound on CNOT gate count when synthesizing a m qubit circuit into n qubits, with $m \leq n$. Data is presented by [13]. The counts for $n = m$ are introduced by [15]. The counts for state preparation ($m = 0$) on two and three qubits are presented by [28], and the count for state preparation on four qubits is introduced by [13]. The generalization and upper bound of is derived by [13]. Note that the CNOT counts grow very fast. For example, the upper bound on any unitary on 10 qubits is about 500,000 CNOT gates.

1) Circuits and Algorithm Equivalence: A quantum transformation can be implemented by multiple distinct quantum circuits. When reasoning in terms of unitaries, there exist multiple decompositions of the unitary into terms that represent gates. Furthermore, when running on hardware, the unitary executed is often subtly different from the intended unitary.

Thus, it is often the case where we want to perform a particular quantum operation A and because of external constraints we end up performing an approximation B, where $B \neq A$. Deciding which algorithm has executed is often referred to as distinguishability and several metrics with operational motivation have been proposed. Trace distance and fidelity [29]–[31] have been proposed for distinguishing states. Metrics such as the diamond norm [32] have been introduced to distinguish processes (algorithms).

Synthesis algorithms use norms to assess the solution quality, and their goal is to minimize $\|U - U_S\|$, where U is the unitary that describes the transformation and U_S is the computed solution. They choose an error threshold ϵ and use it for convergence, $\|U - U_S\| \leq \epsilon$. Early synthesis algorithms use the diamond norm, while more recent efforts [14], [33] use the Hilbert-Schmidt inner product between the conjugate transpose of U and U_s. This is motivated by its lower computational overhead.

$$\langle U, U_s \rangle_{HS} = \mathrm{Tr}(U^\dagger U_s) \tag{3}$$

C. Quantum Processors

Depending on the qubit technology, quantum processors may support different native gate sets, and qubits may be connected in different topologies. We target processors with superconducting qubits since they implement a variety of topologies [2], [18], [19], [34] and are more available. Most offer a native gate set consisting of rotations and CNOT gates $\{R_x(90), R_z(\theta), CNOT\}$. Our results are easily generalized across superconducting qubit architectures which tend to support rotations and a single two qubit gate (CNOT , CRZ or SWAP).

While topology is important for superconducting qubits, implementations using trapped ion [35] qubits provide all-to-all connectivity through Mølmer-Sørensen [36] gates.

III. Synthesis Algorithm

Our goal is to design an algorithm that addresses currently perceived shortcomings of synthesis and that can be easily extended to new hardware in order to enable design space exploration in quantum programming. To be useful during the NISQ device era we use CNOT count as our primary optimality criteria. The synthesis algorithm described in the rest of this section combines a generalized space of parameterized circuits with an approximate A* search [16].

Intuitively, search based synthesis methods rely on the following approach. They start by "enumerating" the space of possible solutions. The construction of this space guarantees that if a solution exists, it will be contained in the enumeration. Then they start walking this space looking for solutions. Previous work uses "randomized" walk through genetic algorithms or Monte-Carlo methods. In contrast, we use a more regimented approach where we formulate the problem as a tree search and can then deploy established algorithms; in our case we use the A* search algorithm. An example of the evolution of a search on a three qubit circuit is depicted in Figure 2.

A. Formulation of Synthesis as a Tree Search Problem

We formulate the problem of synthesis as a tree/graph search problem on circuit structures. The root node of our tree consists of $U3$ gates on every qubit line. For each node in the tree, there is one child for each possible CNOT position, which we can construct by adding a CNOT in that position, followed by two $U3$ gates on the qubit lines affected by the new CNOT.

For any circuit that can be constructed with a finite number of CNOT and $U3$ gates, our tree contains a node that can represent it. We will now provide constructive proof.

As a base case, the empty circuit, which contains 0 gates, implements the identity matrix, which can be represented by the root node with zero for all of its parameters, which also implements the identity. Now, assume that we can represent all circuits with up to i gates. Given a circuit of length $i+1$, we can take the first i gates, and find the node in our tree for it. For the last gate, if it is a CNOT , we can represent it by choosing the child of the node for the first i gates that appends a CNOT in that position, and set the parameters of the two following $U3$ gates to 0. If the last gate is a $U3$, notice that the last gate on every qubit line in our circuit structure for any of our nodes is a $U3$ gate. The root node contains solely $U3$ gates, and any node further down the tree builds on the root node, so no qubit line is empty. The last gate on a qubit line is never a CNOT because we add $U3$ gates immediately after every CNOT . Therefore, the last $U3$ of the $i+1$ circuit is next to a $U3$ gate in the i circuit, and we can combine these two $U3$ gates into a single $U3$ gate with different parameters, and we can use the same node in the tree. We have now found a representation of the circuit of length $i+1$ in our tree.

The gate-set of $U3$ and CNOT is universal for quantum computing, meaning that any unitary matrix can be represented by a circuit consisting of only those gates. Since our tree contains a representation of any such circuit, our tree can represent a circuit that implements any given unitary. Note that a similarly complete tree can be constructed for any universal gateset, such as by using a different multi-qubit gate instead of CNOT in the branching factor of the tree. Since our tree is organized such that circuits with fewer CNOT gates have a lower depth, if we find a lowest depth circuit that implements a given unitary, it will be a solution of lowest CNOT count. We have now reduced the problem of finding a circuit for a given unitary with the lowest CNOT count to a tree search problem, and then the numerical problem of finding values for the parameters. The first problem we can solve via A* search, and the second we can solve using numerical optimizer methods.

B. The Synthesis Algorithm

Our algorithm begins with a target unitary U_{target}, and a target gate-set. It also requires an acceptability threshold ϵ, and a CNOT count limit δ. The threshold ensures the solution is indistinguishable from the target. The CNOT count limit ensures termination and it is selected as depth bounds provided by other competing [13] methods: if we haven't found a solution there are better methods available and we stop. The procedure is described in Algorithms 2 and 1.

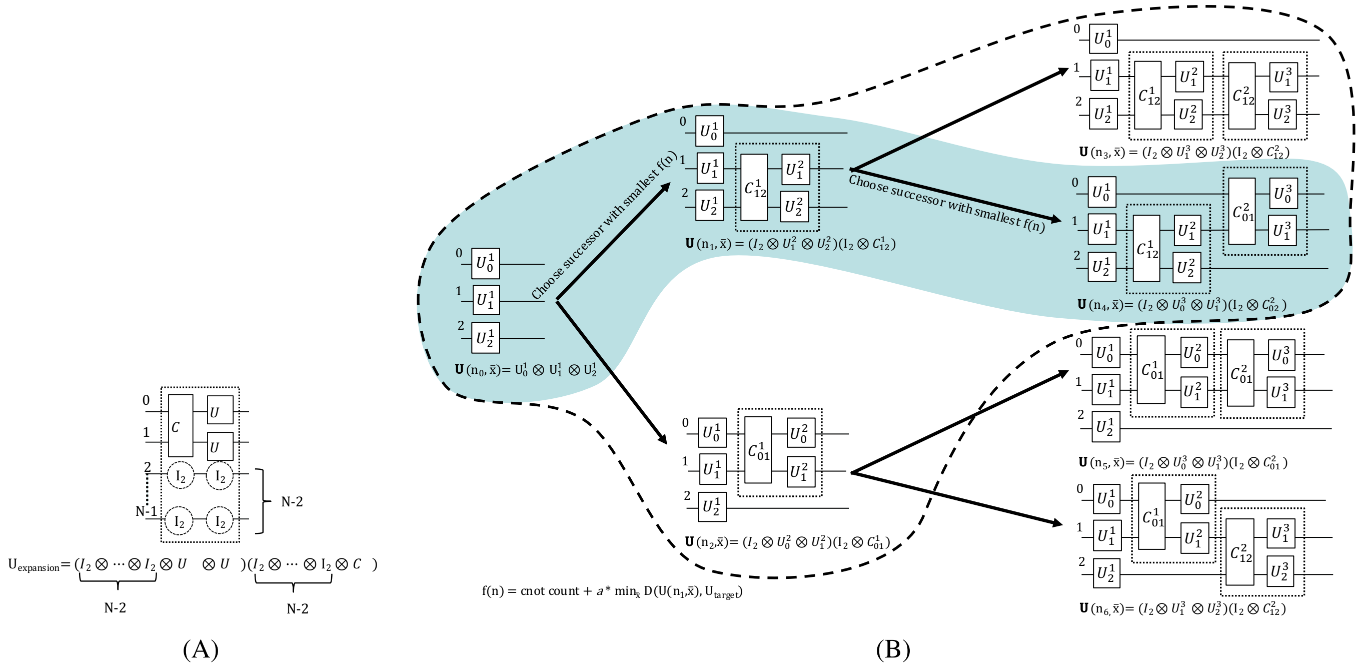
The algorithm relies heavily on a successor function $s(n)$, which takes a node as input and returns a list of nodes, and an optimization function $p(n, U_{target})$, which takes a node and a unitary as input and returns a distance value. The function $h(n)$ is a heuristic function employed by A*, described in the next section.

The successor function, $s(n)$, is defined based on the target gate-set and topology. Given a node n as input, $s(n)$generates child nodes by appending a CNOT followed by two $U3$ gates, using the scheme described in III-A. Generating circuits this way results in minimized CNOT-count at the expense of more single-qubit gates than necessary, which is usually a desirable tradeoff. Single-qubit gate count can be reduced via post processing or by using a different successor function. Different topologies are supported by limiting the positions on which CNOT gates are placed, and other gatesets are supported by replacing CNOT with a different multi-qubit gate.

The optimization function, $p(n, U_{target})$, is used to find the closest matching circuit to a target unitary given a circuit structure. Given a node n and a unitary U_{target}, let $U(n, \overline{x})$ represent the unitary implemented by the circuit structure represented by n when using the vector $\overline{x}$ as parameters for the parameterized gates in the circuit structure. $D(U(n, \overline{x}), U_{target})$ is used as an objective function, and is given to a numerical optimizer, which finds $d = min_{\overline{x}} D(U(n, \overline{x}), U_{target})$. The function $p(n, U_{target})$returns d.

The algorithm begins by generating the root node, which describes a circuit structure with one $U3$ gate on each qubit line. The distance value is found for the root node using $p(n, U_{target})$. These variables are initialized using the root node. The algorithm creates a priority queue that chooses the node n that minimizes $f(n)$, and initializes the queue with the root node as the first entry. Now the algorithm enters a loop, in which it pops nodes from the queue. If no node remains in the queue, the algorithm exits with no solution. Otherwise, a node n is successfully popped from the queue. Its successors n_1-n_k are generated using $s(n)$. For each successor node n_i, the distance $d_i = p(n_i)$ is calculated in parallel. If $d_i < \epsilon$, the current circuit n_i is deemed acceptable and is returned. Otherwise, if the CNOT count of n_i is within the limit δ, the node n_i is pushed onto the priority queue. If there is no acceptable solution with fewer than δ CNOT gates, eventually all possible structures within the given limit will be tried, and no solution will be returned.

The node that is returned from the algorithm, n_{final}, represents a circuit structure that includes a circuit that implements $U_{target}$to a distance within ϵ. To find the specific circuit, the same numerical optimizer can be used, but this time to find $\overline{x}_m = arg\, min_{\overline{x}} D(U(n, \overline{x}), U_{target})$. In practice, it is not necessary to re-run the optimizer since optimizer functions generally return both the minimum value and the values of the parameters that minimize it. The pair of n_{final} and $\overline{x}_m$ constitute a complete description of a quantum circuit, and can be directly converted to quantum assembly.



$\mathrm{U}_{\text{expansion}} = (\underbrace{I_2 \otimes \cdots \otimes I_2}_{N-2} \otimes U \otimes U)(\underbrace{I_2 \otimes \cdots \otimes I_2}_{N-2} \otimes C)$
N-2
Choose successor with smallest f(n)
Choose successor with smallest f(n)
$\mathbf{U}(n_0,\bar{x}) = U_0^1 \otimes U_1^1 \otimes U_2^1$
$\mathbf{U}(n_1,\bar{x}) = (I_2 \otimes U_1^2 \otimes U_2^2)(I_2 \otimes C_{12}^1)$
$\mathbf{U}(n_2,\bar{x}) = (I_2 \otimes U_0^2 \otimes U_1^2)(I_2 \otimes C_{01}^1)$
$\mathbf{U}(n_3,\bar{x}) = (I_2 \otimes U_1^3 \otimes U_2^3)(I_2 \otimes C_{12}^2)$
$\mathbf{U}(n_4,\bar{x}) = (I_2 \otimes U_0^3 \otimes U_1^3)(I_2 \otimes C_{02}^2)$
$\mathbf{U}(n_5,\bar{x}) = (I_2 \otimes U_0^3 \otimes U_1^3)(I_2 \otimes C_{01}^2)$
$\mathbf{U}(n_6,\bar{x}) = (I_2 \otimes U_1^3 \otimes U_2^3)(I_2 \otimes C_{12}^2)$
f(n) = cnot count + a * $\min_{\bar{x}}$ D(U(n$_1$,$\bar{x}$), U$_{\text{target}}$)
(A)
(B)

Fig. 2. *(A) Basic circuit block used for expanding the solution. We generate all alternatives where this structure is placed on linked qubit pairs. Each step adds six additional parameters to the optimization problem. (B) Example evolution of the search algorithm for a three qubit circuit. We start by placing a layer of single qubit gates, then generate the next two possible solutions. Each is evaluated and in this case the upper circuit is closer to the target unitary, leading to a smaller heuristic value. Since this circuit This circuit is then expanded with its possible two successors. These are again instantiated by the optimizer. The second circuit from the top has an acceptable distance and is reported as the solution. The path in blue shows the evolution of the solution. The ansatz circuits enclosed by the dotted line have been evaluated during the search.*

Algorithm 1 Helper Functions
```
1: function S($n$)
2:     return {$n$ + CNOT + $U3 \otimes U3$for all possible CNOT positions}
3:
4: function P($n$, $U$)
5:     return $min_{\overline{x}} D(U(n, \overline{x}), U)$
6:
7: function H($d$)
8:     return $d * a$        ▷ $a$ is a constant determined via experiment. See section 3.3.1
```

Algorithm 2 Search Synthesis
```
1: function SYNTHESIZE($U_{target}$, $\epsilon$, $\delta$)
2:     $n \leftarrow$ representation of $U3$ on each qubit
3:     push $n$ onto $queue$ with priority H($d_{best}$)+0
4:     while $queue$ is not empty do
5:         $n \leftarrow$ pop from $queue$
6:         for all $n_i \in$S($n$) do
7:             $d_i \leftarrow$ P($n_i$, $U_{target}$)
8:             if $d_i < \epsilon$ then
9:                 return $n_i$
10:            if CNOT count of $n_i < \delta$ then
11:                push $n_i$ onto $queue$ with priority H($d_i$)+CNOT count of $n_i$
```

C. A Search Strategy*

The A* algorithm has been developed for graph traversals and it attempts to find a path between a start and target node. At each step, a partial solution is expanded using a successor function, and the successors are added to a priority queue. Then a new partial solution is chosen from the queue that minimizes a cost function. The first path from start to finish is the final solution. Given a partial solution, the algorithm picks the next partial solution based on the cost of its already computed path and an estimate of the cost required to extend it all the way to the target. A* selects the successor node n that minimizes $f(n) = g(n) + h(n)$ where

- $f(n)$ is the estimated total cost of the path from start to finish
- $g(n)$ is the cost of the path from the start to n
- $h(n)$ is a heuristic function that estimates the cost of the cheapest path from n to the target

The algorithm terminates when it reaches the target node or if there are no paths eligible to be extended. The heuristic function is problem-specific and directly determines the time complexity of A*. If the heuristic function is *admissible*, meaning that it never overestimates the actual cost to get to the target, A* is guaranteed to return a least-cost path. A* can be run with an inadmissible heuristic to obtain sub-optimal solutions with a faster runtime than it would take to obtain guaranteed optimal solutions.

For synthesis, the selection of $g(n)$ is obvious as the CNOT count of the partial solution n. The challenge was to determine the heuristic function $h(n)$. After attempts at derivation from first principles, we have opted for a data-driven approach, which is described below.

1) Heuristic Function Tuning: We first use breadth first search for synthesis on training three qubit circuits and examine the details along the final paths . At each partial solution along the path, we recorded the distance value at that step and compared it to the remaining number of CNOT gates (calculated by subtracting the current number of CNOT 's at that step to the final value reach in that run of the program). We then fit the data, and found a best fit line with slope $a = 9.3623$.

The fit gives us the heuristic function $h(n) = D(U(n, \overline{x}_m), U_{target}) * 9.3623$, or $h(n) = p(n, U_{target}) * 9.3623$. Although the fit was not very well correlated ($r^2 = 0.4102$), we found experimentally that the heuristic yielded excellent results. Running the same set of benchmarks with the A* heuristic, we found that the same quality solutions were found, but runtime was significantly faster. For example, brute force search for three qubit QFT takes one hour, while A* takes only seconds.

D. Unitary Distance Metric

We use the following distance function based on the Hilbert-Schmidt inner product. If N is the dimension of the unitaries,

$$D(U, U_{target}) = 1 - \frac{\langle U, U_{target} \rangle_{HS}}{N} = 1 - \frac{\mathrm{Tr}(U^\dagger U_{target})}{N} \quad (4)$$

The formula is based on the fact that the inverse of a unitary matrix is its conjugate transpose. If the synthesis succeeds and U is not distinguishable from U_{target}, the product $U^\dagger U_{target} = I_N$, where I is the identity matrix. Furthermore, the maximum magnitude that the trace of a unitary matrix can have is its size N, which occurs at the identity (up to a phase). The closer $U^\dagger U_{target}$ is to identity, the closer $\frac{\mathrm{Tr}(U^\dagger U_{target})}{N}$ is to N, thus the closer our distance function is to 0.

Note that variations of formulas using Hilbert-Schmidt inner product have been previously used in synthesis algorithms [14], [37], and ours has the following properties

- The distance is 0 when compilation is exact.
- It is fast to compute.
- It has operational meaning.

E. Gradient Computation

Numerical optimizers can find solutions with far fewer objective function calls if they can take advantage of the gradient or Jacobian. We provide a $O(n)$ complexity algorithm to compute the Jacobian of the function described in Equation 4, with implementation details omitted for brevity. The mathematical formulation relies on defining the complex absolute value as $|x| = \sqrt{\mathrm{Re}(x)^2 + \mathrm{Im}(x)^2}$ and reformulating the distance function as described in Equation 5. Equation 6 shows the Jacobian formula.

$$D(U, U_{goal}) = 1 - \left| \sum U_i \cdot U^*_{i_{goal}} \right| \quad (5)$$

$$\frac{dD(U, U_{goal})}{dx} = -\frac{1}{1 - D(U, U_{goal})} \Big(Re(\sum U_i \cdot U^*_{i_{goal}}) \sum Re(U_{i_{goal}}) \frac{dIm(U_i)}{dx} + Im(\sum U_i \cdot U^*_{i_{goal}}) \sum Im(U_{i_{goal}}) \frac{dIm(U_i)}{dx} \Big) \quad (6)$$

IV. Evaluation

Software Implementation: We implemented our algorithm in `Python` 3.7.4, using `numpy` 1.14.4 and in `Rust` 1.44.0. Software is available at `https://github.com/WolfLink/qsearch`. We experiment with the COBYLA, BFGS, and Levenberg-Marquardt numerical optimizers provided with `scipy` 1.2.0. We use `multiprocessing.Pool` for parallelism. Most of the tests ran on a desktop computer with a 4.6GHz Ryzen 9 3900X processor with 12 cores for a total of 24 threads.

Benchmarks: The benchmark suite is composed of third party algorithms used by other evaluation studies [20], [22]. We consider circuits with known optimal implementations, such as Quantum Fourier Transform [38], HHL [39] and important quantum kernels such as Toffoli gate. These allow us to showcase the optimality of our solution. We also consider circuits from domain generators such as the Variational Quantum Eigensolver [40] (VQE) or Transverse Field Ising Models [22], [41]–[43] (TFIM). As no optimal depth circuit is known for these algorithms, they allow us to showcase the benefits of synthesis for circuit optimization. In addition to qubit based circuits we consider the qutrit circuits described in [23].

Experimental Results: Our goal is to demonstrate the value of synthesis to practitioners under several usage scenarios: 1) compiling unitaries; 2) circuit optimization; and 3) gate set design exploration. A summary of the results is presented in Table 3. The columns labeled CNOT show our implementation, annotated with the topology of the target chip. Besides circuit depth, we present the Hilbert-Schmidt distance of the solution and total compilation time.

Customizing for QPU Gate Set and Topology: We target directly the gate set native to the quantum processor. Our initial implementation was tailored for the QNL8QR-v5 processor which supports in hardware the $R_x(90), R_z\theta, CNOT$ gates and its qubits are connected in a line topology. We have also re-targeted the algorithm for the IBM Q 5 qubit chip, with a similar native gate set but a bow-tie/triangle topology.

Use Cases: To showcase the extensibility of the proposed approach we consider synthesis of qutrit gates, a problem of interest to hardware and algorithm designers. To showcase the usability for other algorithmic purposes we consider generating state preparation circuits. To showcase the interaction between synthesis and the rest of the software development stack (optimizing compilers and mappers) we examine using synthesis during the circuit optimization phase. In addition, we are interested in determining the impact of specializing the synthesis algorithm for a different topology. For this, we report the length of the synthesized circuits after being compiled and optimized using QISKit. For example, the "CNOT+QISKit" label describes the experiment where we compile our generated circuit with QISKit.

Comparison with State-of-the-Art: In Table 3, the column labeled UQ shows the number of CNOTs generated by the UniversalQ [21] compiler, a state-of-the-art synthesis tool that uses internally multiple linear algebra based decomposition methods, including Cosine-Sine. For UQ, we report the best result obtained by any decomposition method available.

A. Solution Optimality

In all cases illustrated in Table 3 we were able to synthesize circuits shorter than the theoretical upper bounds provided by [13]: the bound for Q=3 is 20 CNOTs. When comparing against the UniversalQ compiler, we generate significantly shorter circuits, using on average $2.4\times$ fewer CNOTs, and as high as $5.3\times$.

At small qubit count, perhaps the most important comparison is against the length obtained by hand optimization. From this perspective our algorithm behaves well. For example, the optimal CNOT count for Toffoli [44] is six, which our algorithm matches. When mapping to a linear topology, implementations introduce extra SWAPs, up to a total of 12 CNOT gates. Our linear topology Toffoli contains only eight CNOTs. The Fredkin gate is usually implemented as Toffoli sandwiched between two more CNOT gates. Hand optimized Fredkin for linear topologies is available in Cirq [26] with nine CNOTs, while our implementation uses only eight. On a well connected IBM topology we synthesize a Fredkin using only seven CNOTs: IBM QISKit will produce a circuit with eight CNOTs.

The HHL implementation was obtained from the QNL8QR-v5 development team. Mapped to a linear topology by hand, the circuit had seven CNOT gates, while our implementation contains only three.

For QFT3, we match the best known CNOT count of six for the fully connected topology, and improve on the previous best of nine for the linear topology with our circuit of seven CNOTs.

B. Impact of Topology

Embedding the circuit topology within the synthesis algorithm matters, perhaps even more than developing an optimal algorithm for well connected topologies.

The first observation is that existing algorithms report large ($4\times$) proportionality constants when specializing for a restricted topology. In our case we observe only modest increases, up to 15% for the workload and for only five of the tested circuits. In some cases, we obtain circuits shorter than previously known. This indicates that we can handle restricted topologies well.

Even more important is the empirical observation that the rest of the compilation toolkit (circuit optimizers + mappers) can only increase (never decrease) the depth of our synthesized circuits. This is illustrated in Table 3 by the columns with the label "QISKit". In the first experiment, we take the circuits synthesized for a linear topology and compile them with QISKit for the better connected bowtie topology. We enable the highest level of optimization available. The circuits optimized and mapped by QISKit have the same length as the input circuits. In the second experiment, data presented in the Table, we compile the circuits synthesized for the bowtie with QISKit configured for a linear topology. In this case we observe a 53% average increase in CNOT count, with values as high as $4\times$.

To us, this indicates that if the goal for NISQ devices is obtaining optimally short circuits, techniques like ours are more likely to deliver consistently than traditional optimizers and mappers.

C. Synthesis and Circuit Optimization

Optimal implementations for the "EntangledX", TFIM-* and "IBM Challenge" circuits are not available. Thus, they provide an illustration of the benefits of synthesis embedded in the circuit optimization workflow.

The "EntangledX" gate is a building block for a VQE implementation using the [[4,2,2]] error detection code [45] and it is parameterized by a rotation angle. The authors run the circuit iteratively by sampling the parameter for robust behavior, directed by the results from the previous run. Their hand optimized version contains four CNOT gates, which we match for most values of the rotation angle. For some angles, we were able to achieve circuits with only two or three CNOT gates. Note that due to its low execution time of 0.34s, the circuit can be resynthesized in real time during the actual execution on hardware.

TFIM is an exponent of chemical simulations using time dependent Hamiltonians. In this case, domain generators append a fixed function block per time step and circuit depth grows linearly. Domain generators concentrate in reducing "block" depth and can't avoid linear growth. We show circuits generated using the generator described Bassman et al [22] for an increasing number of timesteps up to 60. For all cases we generated circuits with no more than 6 CNOTs, even when the original circuits contained 80+ CNOTs. This amounts to a more than a $10\times$ depth reduction. For comparison, Bassman et al report reductions in circuit depth up to only $\approx 2\times$ when using their domain specific optimizer.

The "IBM Challenge" circuit has been obtained from a recent IBM Qiskit public programming challenge, in which we were able to match the best CNOT count that competitors were able to produce.

D. Retargeting to Qutrits

Qutrits extend qubits to systems with three logical values 0, 1 and 2. They are represented by unitaries from SU(3) and extend from binary to ternary logic to explore a space with 3^n dimensions. There exist several [46], [47] decompositions and parameterizations, all using eight independent parameters. Gates to implement qutrit operations have been explored only recently [23] for qubit based systems, mostly motivated by the need [48] for modeling physical phenomena.

For our study, we implement a CSUM two-qutrit gate, which adds the value of the first qutrit to the second qutrit $CSUM(|11\rangle) = |12\rangle$ and it uses single- and two-qutrit gates. Our synthesis matches the hand optimized implementation by [23]. For brevity, we omit detailed results.

			CNOT count			Mapped by QISKit to linear topology			Unitary distance		Compile time (s)		
ALG	Qubits	Ref	CNOT │	CNOT △	UQ	│	△	UQ	$\|\|$ │ $\|\|$	$\|\|$ △ $\|\|$	T(│)	T(△)	T(UQ)
QFT	3	6	7*	6*	15	7	13	27	$1.33 * 10^{-14}$	$2.22 * 10^{-16}$	1.45	0.99	< 1
Fredkin	3	8	8	7	9	8	16	26	$1.76 * 10^{-14}$	0.0	2.22	4.89	< 1
Toffoli	3	6	8	6	9	8	12	21	$1.14 * 10^{-14}$	0.0	1.47	1.91	< 1
Peres	3	5	7	6	19	7	5	47	$1.13 * 10^{-14}$	$3.33 * 10^{-16}$	0.65	0.75	< 1
HHL	3	N/A	3*	3*	16	3	3	21	$1.25 * 10^{-14}$	0.0	0.30	0.40	< 1
Or	3	6	8	6	10	8	9	19	$1.72 * 10^{-14}$	$4.44 * 10^{-16}$	1.68	2.34	< 1
EntangledX*	3	4	2,3,4	2,3,4	9	4	16	21	$1.26 * 10^{-14}$	$2.22 * 10^{-16}$	0.34	0.50	< 1
TFIM_3_3	3	4	4	4	17	4	4	32	0.0	$2.22 * 10^{-16}$	1.03	0.92	< 1
TFIM_6_3	3	8	6	6	17	6	6	38	$4.44 * 10^{-16}$	$4.44 * 10^{-16}$	3.66	4.84	< 1
TFIM_42_3	3	56	6	6	17	6	7	35	$8.88 * 10^{-16}$	$8.88 * 10^{-16}$	1.65	1.99	< 1
TFIM_60_3	3	80	6	6	17	6	6	41	$6.66 * 10^{-16}$	$8.88 * 10^{-16}$	1.25	1.04	< 1
QFT	4	N/A	14		89	14			$6.66 * 10^{-16}$		56750		< 1
TFIM_30_4	4	60	11		87	11			$9.08 * 10^{-11}$		30135		< 1
IBM Challenge	4	N/A	4		DNR	4			0.0		314		< 1

Fig. 3. *Summary of synthesis results for several algorithms and unitaries The topology used during synthesis is denoted in the caption. Theoretical* CNOT *count upper bound for 3 and 4 qubits are 20 and 100 respectively. *Some gates occasionally resulted in circuits with different CNOT counts due to the optimizers getting stuck in local minima, so the best run out of 10 is listed but the CNOT count was occasionally 1 more for these cases. The gate "EntangledX" is a parameterized gate, and for certain combinations of parameters we were able to produce solutions with fewer CNOTs than the hand-optimized general solution.*

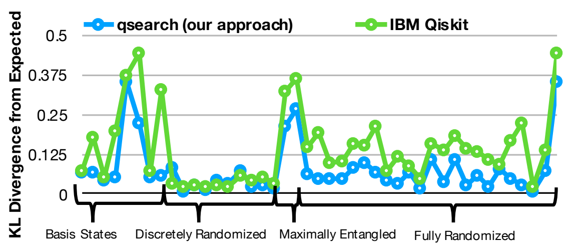


Fig. 4. *KL-divergence comparison for stare preparation circuits.*

E. Synthesis and State Preparation

We have retargeted the algorithm to generate state preparation circuits by changing the distance function to measure the distance from a target state to the state generated by the unitary/circuit performed on the $|0\rangle$ state. We validate the quality of the solution on all basis states and 30 randomly generated states, including two maximally entagled. When compared against the state preparation generator from IBM Qiskit running on the Yorktown chip, we measure an average KL-divergence of $7.1 * 10^{-2}$ and $1.37 * 10^{-1}$ respectively. Besides generating a better quality circuit as illustrated by the differences in the KL-divergence, our circuits are on average 36% shorter than the IBM generated circuits. Figure 4 provides a summary of the results.

F. Acceptability Threshold Tuning

Our algorithm terminates upon finding a circuit with a distance value within an acceptability threshold ϵ. Its value is determined by two requirements: 1) the implementation should be able to meet it in terms of numerical accuracy; and 2) the resulting unitary should be indistinguishable from the original.

For the first criteria we tried synthesizing four of our benchmarks threshold limits at powers of ten ranging from 0.1 to 10^{-12}. We found that with only two exceptions, threshold limits in the range $0.01 \leq \epsilon \leq 10^{-12}$ resulted in final solutions with distance on the order of 10^{-12}–10^{-14}. The two exceptions were both 3-qubit QFT solutions, one with a solution on the order of 10^{-10} and one with 10^{-8}. We concluded that a threshold of 10^{-10} will ensure we have the best quality answer our numerical optimizers will be able to give us.

To ensure this threshold is sufficient for real world applications, we ran another experiment to relate matrix distance to the KL divergence of probability distributions. We generated random unitaries that are close to the identity and multiplied these by fully random unitaries. For each pair of fully random unitary and the product of random unitary and near-identity unitary, we recorded the matrix distance and the KL divergence between the final probability distributions after measuring the result of applying the two unitaries to the same randomly generated state vector, recording the worst case KL divergence after trying 1000 random state vectors. The results showed a clear correlation between KL divergence and Hilbert-Schmidt distance, with the acceptability threshold of 10^{-10} yielding a maximum KL divergence of $2.56 * 10^{-9}$. Even for a looser threshold of 10^{-8}, the maximum KL divergence was $5.20 * 10^{-8}$, so the threshold might even be loosened in practice.

G. Solution Quality

The Hilbert-Schmidt distance between our solution and the original unitary is presented in Table 3. The values range from 10^{-14} to 10^{-17}. We tested the resulting circuits on 1,000 random input state vectors and found that the results are indistinguishable from the target unitary.

We only report the total number of CNOT gates in the generated circuit. The number of parameterized single-qubit gates we generate in a circuit with Q qubits and N CNOTs is $Q + 2 * N$. Generally before running on hardware, each parameterized single-qubit gate is split into five single qubit rotations.

H. Running Time and Scaling

The running time of our algorithm is presented in Table 3. Optimal circuit synthesis is a harshly scaling problem; as the number of qubits increases, the size of unitaries increases exponentially, the branching factor of the search tree increases, and the average circuit depth increases leading to more variables for optimization as well as deeper exploration in the search tree. We employ several approaches to mitigate these scaling factors in order to demonstrate that high-quality search-based synthesis is practical for NISQ circuits.

We are able to mitigate the scaling factors due to longer circuits with more variables for optimization by taking advantage of high-performance numerical optimizers. Better numerical optimizers are able to find solutions with fewer objective function evaluations, reducing the impact of larger unitaries. We have experimented both with derivative-free optimizers CMA-ES [49], COBYLA, and BOBYQA, as well as with gradient-based optimization using the formulae in Section III-E and BFGS and Levenberg-Marquardt. For brevity we omit a detailed comparison but note that the Ceres [50] implementation of Levenberg-Marquardt with gradients provides up to $100\times$ execution time improvements when compared to the implementation of COBYLA provided in scipy, with better scaling as the number of variables increases. In our implementation, the total number of parameters given a circuit with Q qubits and depth d is $Num\_Params = 3*Q+5*d$, and judging by the behavior of QFT circuits synthesized with our algorithm, CNOT count roughly doubles with each additional qubit.

We are able to mitigate the scaling factors due to more difficult search problems by employing smart search algorithms such as A* in order to reduce the number of nodes we evaluate during the search. We have focused on minimal circuit length at the expense of runtime for this paper, but we have been able to perform faster synthesis at the expense of producing longer circuits by employing differently tuned search algorithms. We are also able to mitigate search scaling issues by employing beam search, popping multiple nodes off of the top of the queue at a time, in order to take better advantage of parallelism. Beam searching allows us to evaluate nodes that we would otherwise have to backtrack to in parallel rather than sequentially.

We are further able to mitigate scaling factors by providing a well-optimized implementation. We use a Rust implementation of our matrix computation library which performs up to $10\times$ faster than the original Python implementation. The gate parameterization is minimized by replacing the parameterized single qubit gate after the control line of a CNOT gate with a simpler parameterization with only two parameters (because a parameterized Z gate can commute through the control line of a CNOT and can be absorbed by the parametereized gate on the other side).

V. Discussion

Overall, we believe our results are very encouraging and show the general applicability of quantum circuit synthesis techniques during the NISQ decade(s). Looking back, the field has progressed steadily. Solovay and Kitaev opened the field by showing that a solution exists when using any universal gate set. Later efforts show that solutions exist when restricting the gate sets to "almost native". The emphasis then moved on to reducing the length of produced circuits, and the field has steadily progressed from computing impractically long circuits to computing decent solutions.

We have shown concrete results where we match the shortest known depth for several algorithms, we have shown results where we reduce depth for constrained topologies (line) and we have shown the retargetability of the implementation to new gate sets. Equally important, we have shown empirical evidence that traditional optimization techniques (peephole optimizers and mappers) are unlikely to match the quality of the circuits generated by synthesis. We believe that the results alleviate some of the doubts faced by synthesis approaches: generated circuits are too deep and there is no topology awareness.

Due to its potential, we believe a roadmap for synthesis targeting NISQ devices is worth developing. For practical purposes, it is essential to produce circuits short enough to run on hardware with acceptable error. It is also important for synthesis runtime to scale reasonably. Given that we have shown optimality and topology awareness, for the near future, scalability at small qubit scale is worth exploring as it will lead to establishing robust building blocks for approaches targeting larger number of qubits.

Synthesis for early NISQ (small) circuits:

- *Better numerical optimizers.* The judicious choice of the numerical optimizer is probably the most important factor. The second step is to employ meta-optimization techniques, such as multi-start techniques. It is also worth considering building ad-hoc optimizers for synthesis based on tensor networks and gradient descent. These have the advantage of high GPU performance.
- *Better parallelization of the search algorithm.* Currently we leverage two levels of parallelism: BLAS for parallelism within matrix computations, and multiple processes for simultaneous invocations of the numerical optimizer. Currently, our matrices are too small to take full advantage of BLAS, and while we use beams to increase the number of ansatz circuits we optimize in parallel, this improvement has diminishing returns. We expect both of these avenues of parallelism to become more useful with larger circuits.

Synthesis for late NISQ (large) circuits: For circuits with tens of qubits memory and computational requirements for synthesis may be prohibitive, as unitaries scale exponentially with 2^q. Given an already existing circuit, a straightforward way to incorporate synthesis is to partition it in manageably sized blocks, optimize these individually, and recombine. For algorithm discovery, synthesis will have to be incorporated into generative models for domain science.

VI. Related Work

A fundamental result, which spurred the apparition of quantum circuit synthesis is provided by the Solovay Kitaev (SK) theorem. The theorem relates circuit depth to the quality of the approximation and its proof is by construction [4]–[6]. Different approaches [4], [7]–[9], [51]–[57] to synthesis have been introduced since, with the goal of generating shorter depth circuits. These can be coarsely classified based on several criteria: 1) target gate set; 2) algorithmic approach; and 3) solution distinguishability.

Target Gate Set: The SK algorithm is applicable to any universal gate set. Later examples include synthesis of z-rotation unitaries with Clifford+V approximation [58] or Clifford+T gates [59]. When ancillary qubits are allowed, one can synthesize single qubit unitaries with the Clifford+T gate set [59]–[61]. While these efforts propelled the field of synthesis, they are not used on NISQ devices, which offer a different gate set ($R_x, R_z, CNOT$ and Mølmer-Sørensen all-to-all). Several [13], [15], [37] other algorithms, discussed below have since emerged.

Algorithmic Approaches: The earlier attempts inspired by Solovay Kitaev use a recursive (or divide-and-conquer) formulation, sometimes supplemented with search heuristics at the bottom. More recent search based approaches are illustrated by the Meet-in-the-Middle [8] algorithm.

Several approaches use techniques from linear algebra for unitary/tensor decomposition. [53] use QR matrix factorization via Given's rotation and Householder transformation [54], but there are open questions as to the suitability for hardware implementation because these algorithms are expressed in terms of row and column updates of a matrix rather than in terms of qubits.

The state-of-the-art upper bounds on circuit depth are provided by techniques [13], [15] that use Cosine-Sine decomposition. The Cosine-Sine decomposition was first used by [62] for compilation purposes. In practice, commercial compilers ubiquitously deploy only KAK [12] decompositions for two qubit unitaries.

The basic formulation of these techniques is topology independent. Specializing for topology increases the upper bound on circuit depth by large constants; [15] mention a factor of nine, improved by [13] to $4\times$. The published approaches are hard to extend to different qubit gate sets and it remains to be seen if they can handle qutrits[1]. Furthermore, the numerical techniques [63] required for CSD still require refinements as they cannot handle numerically challenging cases.

Several techniques use numerical optimization, much as we did. They describe the gates in their variational/continuous representation and use optimizers and search to find a gate decomposition and instantiation. The work closest to ours is [37] which uses numerical optimization and brute force search to synthesize circuits for a processor using trapped ion qubits. Their main advantage is the existence of all-to-all Mølmer-Sørensen gates, which allow a topology independent approach. The main difference between our work and theirs is that they use randomization and genetic algorithms to search the solution space, while we show a more regimented way. When Martinez et al. describe their results, they claim that Mølmer-Sørensen counts are directly comparable to CNOT counts. By this metric, we seem to generate comparable or shorter circuits than theirs. It is not clear how their approach behaves when topology constraints are present. The direct comparison is further limited due to the fact that they consider only randomly generated unitaries, rather than algorithms or well understood gates such as Toffoli or Fredkin.

Another topology independent numerical optimization technique is presented by [14]. The main contribution is to use a quantum annealer to do searches over sequences of increasing gate depth. They report results only for two qubit circuits.

All existing studies focus on the quality of the solution, rather than synthesis speed. They also report results for low qubit concurrency: Khatri et al. [14] for two qubit systems, Martinez et al. [37] for systems up to four qubits.

Solution Distinguishability: Synthesis algorithms can be classified as exact or approximate based on distinguishability. This is a subtle classification criteria, as many algorithms can be viewed as either. For example, [8] proposed a divide-and-conquer algorithm called Meet-in-the-Middle (MIM). Designed for exact circuit synthesis, the algorithm may also be used to construct an ϵ-approximate circuit. The results seem to indicate that the algorithm failed to synthesize a three qubit QFT circuit. Furthermore, on NISQ devices, the target gate set of the algorithm (e.g. T gate) may be itself implemented as an approximation when using native gates. We classify our implementation as approximate since we rely on numerical optimization and therefore must accept solutions at a small distance from the original unitary.

VII. Conclusion

In this work we have shown methods to compile arbitrary quantum unitaries into a sequence of gates native to several superconducting qubit based architectures. The algorithm we develop is topology aware and easily re-targeted to new gates sets or topologies. Results indicate that we can match, or even improve on the shortest depth circuit implementation published for several widely used algorithms and gates, especially when topology is restrictive. We also show empirical evidence which supports an important conjecture: the benefits of incorporating topology directly into synthesis cannot be replicated if relying on all-to-all synthesis and traditional (peephole base) optimizing quantum compilers or mappers.

The method is slow but produces good results in practice. For the early NISQ era, which is likely to be characterized by hero experiments, the overhead seems acceptable. Even when superseded by faster algorithms, we believe our results provide a good quality measure threshold for these implementations.

Looking forward, better numerical optimizers would enhance the palatability of quantum circuit synthesis by alleviating some of the need for developing better search algorithms. Additionally, search algorithm development will need to balance runtime with circuit length optimality.

Acknowledgements

This work was supported by the testbed program of the Advanced Scientific Computing Research for Basic Energy Sciences program, Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

[1] [46] describes a method using Givens rotations and Householder decomposition.

References

[1] S. K. Moore, "Ibm edges closer to quantum supremacy with 50-qubit processor." https://spectrum.ieee.org/tech-talk/computing/hardware/ibm-edges-closer-to-quantum-supremacy-with-50qubit-processor, 2017.

[2] J. Kelly, "A preview of bristlecone, google's new quantum processor." https://research.googleblog.com/2018/03/a-preview-of-bristlecone-googles-new.html, 2018.

[3] J. Hsu, "Ces 2018: Intel's 49-qubit chip shoots for quantum supremacy." https://spectrum.ieee.org/tech-talk/computing/hardware/intels-49qubit-chip-aims-for-quantum-supremacy, 2018.

[4] C. M. Dawson and M. A. Nielson, "The Solovay-Kitaev Algorithm," *Quant. Info. Comput.*, vol. 6, no. 1, pp. 81–95, 2005.

[5] A. B. Nagy, "On an implementation of the Solovay-Kitaev algorithm," *arXiv:quant-ph/0606077*, 2016.

[6] O. Al-Ta'Ani, *Quantum Circuit Synthesis using Solovay-Kitaev Algorithm and Optimization Techniques*. PhD thesis, Kansas State University, 2015.

[7] A. De Vos and S. De Baerdemacker, "Block-zxz synthesis of an arbitrary quantum circuit," *Phys. Rev. A*, vol. 94, p. 052317, Nov 2016.

[8] M. Amy, D. Maslov, M. Mosca, and M. Roetteler, "A meet-in-the-middle algorithm for fast synthesis of depth-optimal quantum circuits," *Trans. Comp.-Aided Des. Integ. Cir. Sys.*, vol. 32, pp. 818–830, June 2013.

[9] E. A. Martinez, T. Monz, D. Nigg, P. Schindler, and R. Blatt, "Compiling quantum algorithms for architectures with multi-qubit gates," *New Journal of Physics*, vol. 18, no. 6, p. 063029, 2016.

[10] V. Kliuchnikov, D. Maslov, and M. Mosca, "Fast and efficient exact synthesis of single-qubit unitaries generated by clifford and t gates," *Quantum Info. Comput.*, vol. 13, pp. 607–630, July 2013.

[11] B. Giles and P. Selinger, "Exact synthesis of multiqubit clifford+t circuits," *Phys. Rev. A*, vol. 87, p. 032332, Mar 2013.

[12] R. R. Tucci, "An Introduction to Cartan's KAK Decomposition for QC Programmers," *arXiv e-prints*, pp. quant–ph/0507171, Jul 2005.

[13] R. Iten, R. Colbeck, I. Kukuljan, J. Home, and M. Christandl, "Quantum circuits for isometries," *Physical Review A*, vol. 93, p. 032318, Mar 2016.

[14] S. Khatri, R. LaRose, A. Poremba, L. Cincio, A. T. Sornborger, and P. J. Coles, "Quantum-assisted quantum compiling," *arXiv e-prints*, p. arXiv:1807.00800, Jul 2018.

[15] V. V. Shende, S. S. Bullock, and I. L. Markov, "Synthesis of quantum-logic circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, pp. 1000–1010, June 2006.

[16] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, pp. 100–107, July 1968.

[17] M. J. D. Powell, "A direct search optimization method that models the objective and constraint functions by linear interpolation," in *Advances in Optimization and Numerical Analysis* (S. Gomez and J.-P. Hennart, eds.), pp. 51–67, Dordrecht: Springer Netherlands, 1994.

[18] I. Siddiqi, ""quantum nanoelectronics laboratory, university of california at berkeley"." http://qnl.berkeley.edu/, 2019.

[19] IBM, "Ibm q 5 yorktown chip.." https://www.research.ibm.com/ibm-q/technology/devices/#ibmqx2, 2019.

[20] P. Murali, J. M. Baker, A. Javadi Abhari, F. T. Chong, and M. Martonosi, "Noise-Adaptive Compiler Mappings for Noisy Intermediate-Scale Quantum Computers," *arXiv e-prints*, p. arXiv:1901.11054, Jan 2019.

[21] R. Iten, O. Reardon-Smith, L. Mondada, E. Redmond, R. Singh Kohli, and R. Colbeck, "Introduction to UniversalQCompiler," *arXiv e-prints*, p. arXiv:1904.01072, Apr 2019.

[22] L. Bassman, S. Gulania, C. Powers, R. Li, T. Linker, K. Liu, T. K. S. Kumar, R. K. Kalia, A. Nakano, and P. Vashishta, "Domain-specific compilers for dynamic simulations of quantum materials on quantum computers," 2020.

[23] M. Blok, V. Ramasesh, J. Colless, K. O'Brien, T. Schuster, N. Yao, and I. Siddiqi., "Implementation and applications of two qutrit gates in superconducting transmon qubits.." Bulletin of the American Physical Society 2018, 2018.

[24] D. Deutsch, "Quantum computational networks," in *Proceedings of The Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 425, pp. 73–90, The Royal Society Publishing, 09 1989.

[25] "Ibm qiskit." Available at https://qiskit.org/.

[26] "Google cirq." Available at https://github.com/quantumlib/Cirq.

[27] Rigetti., "Forest and pyquil documentation.." http://docs.rigetti.com/en/stable/.

[28] M. Znidaric, O. Giraud, and B. Georgeot, "Optimal number of controlled-not gates to generate a three-qubit state," *Phys. Rev. A*, vol. 77, 03 2008.

[29] A. Gilchrist, N. K. Langford, and M. A. Nielsen, "Distance measures to compare real and ideal quantum processes," *Phys. Rev. A*, vol. 71, p. 062310, Jun 2005.

[30] H.-P. Breuer, E.-M. Laine, and J. Piilo, "Measure for the Degree of Non-Markovian Behavior of Quantum Processes in Open Systems," *Physical Review Letters*, vol. 103, p. 210401, Nov 2009.

[31] M. A. Nielsen and I. L. Chuang, *Frontmatter*, pp. i–viii. Cambridge University Press, 2010.

[32] A. Y. Kitaev, A. H. Shen, and M. N. Vyalyi, *Classical and Quantum Computation*. Boston, MA, USA: American Mathematical Society, 2002.

[33] V. Kliuchnikov, A. Bocharov, and K. M. Svore, "Asymptotically Optimal Topological Quantum Compiling," *Physical Review Letters*, vol. 112, p. 140504, Apr 2014.

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

[35] J. I. Cirac and P. Zoller, "Quantum computations with cold trapped ions," *Phys. Rev. Lett.*, vol. 74, pp. 4091–4094, May 1995.

[36] A. Sørensen and K. Mølmer, "Entanglement and quantum computation with ions in thermal motion," *Physical Review A*, vol. 62, p. 022311, Aug 2000.

[37] E. Martinez, T. Monz, D. Nigg, P. Schindler, and R. Blatt, "Compiling quantum algorithms for architectures with multi-qubit gates," *ArXiv e-prints*, July 2016.

[38] V. NAMIAS, "The Fractional Order Fourier Transform and its Application to Quantum Mechanics," *IMA Journal of Applied Mathematics*, vol. 25, pp. 241–265, 03 1980.

[39] A. W. Harrow, A. Hassidim, and S. Lloyd, "Quantum Algorithm for Linear Systems of Equations," *Physical Review Letters*, vol. 103, p. 150502, Oct. 2009.

[40] J. R. McClean, J. Romero, R. Babbush, and A. Aspuru-Guzik, "The theory of variational hybrid quantum-classical algorithms," *New Journal of Physics*, vol. 18, no. 2, p. 23023, 2016.

[41] S. Suzuki, J.-i. Inoue, and B. K. Chakrabarti, *Quantum Ising Phases and Transitions in Transverse Ising Models*, vol. 859. Moskovski Inzhenerno-Fisitscheski Institute, 2013.

[42] D. Shin, H. Hübener, U. De Giovannini, H. Jin, A. Rubio, and N. Park, "Phonon-driven spin-floquet magneto-valleytronics in mos2," *Nature Communications*, vol. 9, no. 1, p. 638, 2018.

[43] L. Bassman, K. Liu, Y. Geng, D. Shebib, A. Krishnamoorthy, and P. Vashishta, "Simulating dynamic material properties on near-term quantum computers." Bulletin of the American Physical Society, 2020.

[44] V. V. Shende and I. L. Markov, "On the CNOT-cost of TOFFOLI gates," *arXiv e-prints*, p. arXiv:0803.2316, Mar 2008.

[45] M. Grassl, T. Beth, and T. Pellizzari, "Codes for the quantum erasure channel," *Phys. Rev. A*, vol. 56, pp. 33–38, Jul 1997.

[46] N. Vitanov, "Synthesis of arbitrary su(3) transformations of atomic qutrits," *Phys. Rev. A*, vol. 85, 03 2012.

[47] J. B. Bronzan, "Parametrization of su(3)," *Phys. Rev. D*, vol. 38, pp. 1994–1999, Sep 1988.

[48] K. A. Landsman, C. Figgatt, T. Schuster, N. M. Linke, B. Yoshida, N. Y. Yao, and C. Monroe, "Verified quantum information scrambling," *Nature*, vol. 567, no. 7746, pp. 61–65, 2019.

[49] N. Hansen, "The CMA evolution strategy: A tutorial," *CoRR*, vol. abs/1604.00772, 2016.

[50] S. Agarwal, K. Mierle, and Others, "Ceres solver." http://ceres-solver.org.

[51] A. Bocharov and K. M. Svore, "Resource-optimal single-qubit quantum circuits," *Phys. Rev. Lett.*, vol. 109, p. 190501, Nov 2012.

[52] B. Giles and P. Selinger, "Exact synthesis of multiqubit Clifford+T circuits," *Physical Review Letters.*, vol. 87, p. 032332, Mar. 2013.

[53] S. S. Bullock and I. L. Markov, "An arbitrary two-qubit computation in 23 elementary gates or less," in *Proceedings 2003. Design Automation Conference (IEEE Cat. No.03CH37451)*, pp. 324–329, June 2003.

[54] J. Urias, "Householder factorization of unitary matrices," *J. Mathematical Physics*, vol. 51, p. 072204, 2010.

[55] M. Möttönen, J. J. Vartiainen, V. Bergholm, and M. M. Salomaa, "Quantum circuits for general multiqubit gates," *Phys. Rev. Lett.*, vol. 93, p. 130502, Sep 2004.

[56] M. Amy and M. Mosca, "T-count optimization and Reed-Muller codes," *arXiv:1601.07363v1*, 2016.

[57] G. Seroussi and A. Lempel, "Factorization of symmetric matrices and trace-orthogonal bases in finite fields," *SIAM Journal on Computing*, vol. 9, no. 4, pp. 758–767, 1980.

[58] N. J. Ross, "Optimal ancilla-free clifford+v approximation of z-rotations," *Quantum Info. Comput.*, vol. 15, pp. 932–950, Sept. 2015.

[59] V. Kliuchnikov, D. Maslov, and M. Mosca, "Practical approximation of single-qubit unitaries by single-qubit quantum clifford and t circuits," *IEEE Transactions on Computers*, vol. 65, pp. 161–172, Jan 2016.

[60] A. Kitaev, A. Shen, and M. Vyalyi, *Classical and Quantum Computation*. Boston, MA: American Mathematical Society, 2012.

[61] A. Paetznick and K. M. Svore, "Repeat-until-success: Non-deterministic decomposition of single-qubit unitaries," *Quantum Info. Comput.*, vol. 14, pp. 1277–1301, Nov. 2014.

[62] R. R. Tucci, "A Rudimentary Quantum Compiler(2cnd Ed.)," *arXiv e-prints*, pp. quant–ph/9902062, Feb 1999.

[63] B. D. Sutton, "Computing the complete cs decomposition," *Numerical Algorithms*, vol. 50, no. 1, pp. 33–65, 2009.