LEAP: Scaling Numerical Optimization Based Synthesis Using an Incremental Approach

ETHAN SMITH and MARC GRAU DAVIS, University of California, Berkeley
JEFFREY LARSON, Argonne National Laboratory
ED YOUNIS, LINDSAY BASSMAN OFTELIE, WIM LAVRIJSEN, and COSTIN IANCU, Lawrence Berkeley National Laboratory

While showing great promise, circuit synthesis techniques that combine numerical optimization with search over circuit structures face scalability challenges due to a large number of parameters, exponential search spaces, and complex objective functions. The LEAP algorithm improves scaling across these dimensions using iterative circuit synthesis, incremental reoptimization, dimensionality reduction, and improved numerical optimization. LEAP draws on the design of the optimal synthesis algorithm QSearch by extending it with an incremental approach to determine constant prefix solutions for a circuit. By narrowing the search space, LEAP improves scalability from four to six qubit circuits. LEAP was evaluated with known quantum circuits such as QFT and physical simulation circuits like the VQE, TFIM, and QITE. LEAP can compile four qubit unitaries up to $59 \times$ faster than QSearch and five and six qubit unitaries with up to $1.2 \times$ fewer CNOTs compared to the QFAST package. LEAP can reduce the CNOT count by up to $36 \times$, or $7 \times$ on average, compared to the CQC Tk compiler. Despite its heuristics, LEAP has generated optimal circuits for many test cases with a priori known solutions. The techniques introduced by LEAP are applicable to other numerical optimization based synthesis approaches.

CCS Concepts: • Hardware → Quantum computation; • Computer systems organization → Quantum computing;

Additional Key Words and Phrases: Gate-based quantum computing, Quantum circuit synthesis

ACM Reference format:
Ethan Smith, Marc Grau Davis, Jeffrey Larson, Ed Younis, Lindsay Bassman Oftelie, Wim Lavrijsen, and Costin Iancu. 2023. LEAP: Scaling Numerical Optimization Based Synthesis Using an Incremental Approach. ACM Trans. Quantum Comput. 4, 1, Article 5 (February 2023), 23 pages.
https://doi.org/10.1145/3548693

1 INTRODUCTION
Quantum synthesis techniques generate circuits from high-level mathematical descriptions of an algorithm. They can provide a powerful tool for circuit optimization, hardware design exploration,
and algorithm discovery. An important quality metric of synthesis, and of compilers in general, is circuit depth, which relates directly to the program performance on hardware. Short-depth circuits are especially important for noisy intermediate-scale quantum (NISQ)-era devices, characterized by limited coherence time and noisy gates. Here, synthesis provides a critical capability in enabling experimentation where only the shortest depth circuits provide usable outputs. Furthermore, synthesizing short-depth circuits is a powerful building block useful in circuit partitioning algorithms that can be used to optimize circuits with hundreds of qubits. Reducing the depth of each block can greatly reduce the overall depth of the partitioned circuit.

In general, two concepts are important when thinking about synthesis algorithms: circuit structure captures the application of gates on “physical” qubit links, whereas function captures the gate operations, such as rotation angle $R_z(\theta)$. Recently introduced techniques can generate short-depth circuits in a topology-aware manner by combining numerical optimization of parameterized gate representations (e.g., $U_3$) to determine function together with search over circuit structures. Regarding circuit depth, their efficacy surpasses that of traditional optimizing compilers such as IBM Qiskit and CQC Tket, or of other available synthesis tools such as UniversalQ.\(^1\)

An exemplar of synthesis approaches is QSearch, which provides optimal-depth synthesis and has been shown to match known optimal quantum algorithm implementations for circuits such as QFT. QSearch grows a circuit by adding layers of parameterized gates and permuting gate placement at each link, building on the previous best placements to form a circuit structure. A numerical optimizer is run on each candidate circuit structure to instantiate the function that “minimizes” a score (distance from the target unitary based on the Hilbert–Schmidt norm). This score guides the A* search algorithm to extend and evaluate the next partial solution. Other numerical optimization based synthesis algorithms use a similar approach.

While providing good-quality results, however, these techniques face scalability challenges: (1) the number of parameters to optimize grows with circuit depth; (2) the number of intermediate solutions to consider is exponential; and (3) the objective function for optimization is complex, and optimizers may get stuck in local minima. LEAP (Larger Exploration by Approximate Prefixes) has been designed to improve the scalability of QSearch, and it introduces several novel techniques directly extensible to the broader class of search or numerical optimization based synthesis.

Prefix circuit synthesis. Designed to improve scaling, LEAP prunes the search space by limiting backtracking depth and by coarsening the granularity of the backtrack steps. Our branch-and-bound algorithm monitors progress during search and employs “execution-driven” heuristics to decide which partial solutions are good prefix candidates for the final solution. Whenever a prefix is chosen, the question is whether to reuse the structure (gate placement) or structure and function (gate instantiation) together. The former approach prunes the search space, whereas the latter prunes both the search and parameter spaces.

Incremental resynthesis. The end result of incremental prefix synthesis (or other divide-and-conquer methods, partitioning techniques, etc.) is that circuit pieces are processed in disjunction, with the potential of missing the global optimum. Intuitively, LEAP gravitates toward the solution by combining local optimization on disjoint subcircuits. By chopping and combining pieces of the final circuit, we can create new, unseen subcircuits for the optimization process. Overall, this technique is designed to improve the solution quality for any divide-and-conquer or other hierarchical approach.

---

\(^1\)The UniversalQ algorithms have been recently incorporated into IBM Qiskit. For brevity, in the rest of this article we will refer to it as Qiskit-synth.
**Dimensionality reduction.** This technique could improve both scalability and solution quality. QSearch and LEAP require sets of gates that can fully describe the Hilbert subspace explored by the input transformation. This approach ensures convergence, but in many cases it may overfit the problem. We provide an algorithm to delete any parameterized gates that do not contribute to the solution, thereby reducing the dimension of the optimization problems. When applied directly to the final solution, dimensionality reduction may improve the solution quality by deleting single-qubit gates. Dimensionality reduction may also be applied in conjunction with prefix circuit synthesis, improving both scalability and solution quality.

**Multistart numerical optimization.** This technique affects both scalability and the quality of the solution. Any stand-alone numerical optimizer is likely to have a low success rate when applied to problem formulations that involve quantum circuit parameterizations. Multistart optimization [13] improves on the success rate and quality of solution (avoids local minima) by running multiple numerical optimizations in conjunction. Each individual multioptimizer step may become slower, but improved solutions may reduce the chance of missing an optimal solution, causing further search expansion.

LEAP has been implemented as an extension to QSearch, and it has been evaluated on traditional “gates” such as mul and adder, as well as full-fledged algorithms such as Quantum Fourier Transformation (QFT) [11], HLF [14], Variational Quantum Eigensolver (VQE) [15], Transverse Field Ising Model (TFIM) [16, 17], and Quantum Imaginary Time Evolution (QITE) [18]. We compare its behavior with state-of-the-art synthesis approaches: QSearch, QFAST [7], Tket [9], and Qiskit-synth [10]. QSearch scales up to four qubits, whereas LEAP can compile four-qubit unitaries up to $59\times$ faster than QSearch and scales up to six qubits. On well-known quantum circuits such as VQE, QFT, and physical simulation circuits such as TFIM, LEAP with resynthesis can reduce the CNOT count by up to $48\times$, or $11\times$ on average. Our heuristics rarely affect solution quality, and LEAP can frequently match optimal-depth solutions. At five and six qubits, LEAP synthesizes circuits with up to $1.19\times$ fewer CNOTs on average compared with QFAST, albeit with an average $3.55\times$ performance penalty. LEAP can be one order of magnitude slower than Qiskit-synth while providing two or more orders of magnitude shorter circuits. Compared with Tket, LEAP reduces the depth on average by $7.70\times$ while taking significantly longer in runtime.

All of our techniques affect behavior and performance in a nontrivial way:

- Compared with QSearch, prefix synthesis reduces by orders of magnitude the number of partial solutions explored, leading to significant speedup.

- Incremental resynthesis reduces circuit depth by 15% on average, albeit with large increases in running time.

- Dimensionality reduction eliminates up to 40% of $U_3$ gates on average (thus reducing the number of parameters) and shortens the circuit critical path.

- Multistart increases the optimizer success rate from 15% (best value observed for any stand-alone optimizer) to 99%. For a single optimization run, however, multistart is up to $10\times$ slower than the underlying numerical optimizer.

Overall, we believe that LEAP provides a very competitive circuit optimizer for circuits on NISQ devices up to six qubits. We believe that our techniques can be easily generalized or transferred directly to other algorithms based on the search of circuit structures or numerical optimization. For example, resynthesis, dimensionality reduction, and multistart optimization are directly applicable to QFAST, and resynthesis is applicable to Qiskit-synth. We can expect that synthesis techniques using divide-and-conquer or partitioning methods will be mandatory for scalability to the number of qubits (in thousands) provided by future near-term processors. Our techniques provide valuable information to these budding approaches.
The rest of this article is structured as follows. In Section 2, we describe the problem and its challenges. The proposed solutions are discussed in Sections 3 through 6. The experimental evaluation is presented in Section 7. In Section 9, we discuss the implications of our approach. Related work is presented in Section 10. In Section 11, we briefly summarize our conclusions.

2 BACKGROUND

In quantum computing, a qubit is the basic unit of quantum information. The 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 $|0\rangle = (1, 0)$ and $|1\rangle = (0, 1)$.

The generic qubit state is a superposition of the basis states, namely $|\psi\rangle = \alpha |0\rangle + \beta |1\rangle$, with complex amplitudes $\alpha$ and $\beta$, such that $|\alpha|^2 + |\beta|^2 = 1$.

The prevalent model of quantum computation is the circuit model introduced in the work of Deutsch [19], 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—that is, $UU^* = U^*U = I$.

In the circuit model, a single-qubit gate is represented by a $2 \times 2$ 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.

$$U_3(\theta, \phi, \lambda) = \begin{pmatrix} \cos \frac{\theta}{2} & -e^{i\lambda} \sin \frac{\theta}{2} \\ e^{i\phi} \sin \frac{\theta}{2} & e^{i\lambda+i\phi} \cos \frac{\theta}{2} \end{pmatrix}$$

A quantum transformation (algorithm, circuit) on $n$ qubits is represented by a unitary matrix $U$ of size $2^n \times 2^n$. 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.

Circuit synthesis. The goal of circuit synthesis is to decompose unitaries from $SU(n)$ into a product of terms, where each individual term (e.g., from $SU(2)$ and $SU(4)$) 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 resulting circuit and by the solution distinguishability from the original unitary.

Circuit length provides one of the main optimality criteria for synthesis algorithms: shorter circuits are better. The CNOT count is a direct indicator of overall circuit length, since the number of single-qubit generic gates introduced in the circuit is proportional to a constant given by decomposition (e.g., $ZXZXZ$) rules. Since CNOT gates have low fidelity on NISQ devices, state-of-the-art approaches [1, 2] directly attempt to minimize their count. Longer-term, single-qubit gate count (and circuit critical path) is likely to augment the quality metric for synthesis.

Synthesis algorithms use distance metrics to assess the solution quality. 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 $\epsilon$ and use it for convergence, $\|U - U_S\| \leq \epsilon$. Early synthesis algorithms used the diamond norm, whereas more recent efforts [4, 20] use a metric based on the Hilbert–Schmidt inner product between $U$ and $U_S$.

$$\langle U, U_S \rangle_{HS} = Tr(U^\dagger U_S)$$

This is motivated by its lower computational overhead.
Fig. 1. Unitaries (above) and tensor products (below). The unitary $U$ represents a $n = 3$ qubit transformation, where $U$ is a $2^3 \times 2^3$ matrix. The unitary is implemented (equivalent or approximated) by the circuit on the right-hand side. The single-qubit unitaries are $2 \times 2$ matrices, whereas CNOT is a $2^2 \times 2^2$ matrix. The computation performed by the circuit is $(I_2 \otimes U_4 \otimes U_5)(I_2 \otimes \text{CNOT})(U_1 \otimes U_2 \otimes U_3)$, where $I_2$ is the identity $2 \times 2$ matrix and $\otimes$ is the tensor product operator. The right-hand side shows the tensor product of $2 \times 2$ matrices.

2.1 Optimal-Depth Topology-Aware Synthesis

QSearch [6] introduces an optimal-depth topology-aware synthesis algorithm that has been demonstrated to be extensible across native gate sets (e.g., $\{RX, RZ, \text{CNOT}\}$, $\{RX, RZ, CZ\}$) and to multilevel systems such as qutrits.

The approach employed in QSearch is canonical for the operation of other synthesis approaches that employ numerical optimization. Conceptually, the problem can be thought of as a search over a tree of possible circuit structures containing parameterized gates. A search algorithm provides a principled way to walk the tree and evaluate candidate solutions. For each candidate, a numerical optimizer instantiates the function (parameters) of each gate to minimize some distance objective function.

QSearch works by extending the circuit structure a layer at a time. At each step, the algorithm places a two-qubit expansion operator in all legal placements. The operator contains one CNOT gate and two $U_3(\theta, \phi, \lambda)$ gates. QSearch then evaluates these candidates using numerical optimization to instantiate all single-qubit gates in the structure. An $A^*$ [12] heuristic determines which of the candidates is selected for another layer expansion, as well as the destination of the backtracking steps. Figure 2 illustrates this process for a three-qubit circuit.

Although theoretically able to solve for any “program” (unitary) size, the scalability of QSearch is limited in practice to four-qubit programs because of several factors. The $A^*$ strategy determines the number of solutions evaluated: at best, this is linear in depth; at worst, it is exponential. Any technique to reduce the number of candidates, especially when deep, is likely to improve performance. Our prefix synthesis solution is discussed in Section 3.

Since each expansion operator has two $U_3$ gates, accounting for six\(^2\) parameters, circuit parameterization grows linearly with depth. Numerical optimizers scale at best with a high-degree polynomial in the number of parameters, making optimization of long circuits challenging. Any technique to reduce the number of parameters is likely to improve performance. Dimensionality reduction is discussed further in Section 5.

The scalability and the quality of the numerical optimizer matter. Faster optimizers are desirable, but their quality affects performance nontrivially. Our experimentation with CMA-ES [21], L-BFGS [22], and Google Ceres [23] shows that the QSearch success rate of obtaining a solution from a valid structure can vary from 20% to 1% for longer circuits. Besides this measurable outcome, the propensity of optimizers to get stuck in local minima and plateaus can have a negative effect on scalability by altering the search path. A more nuanced approach to optimization and\(^\text{2}\)In practice, QSearch uses five parameters because of commutativity rules between $RZ(\theta)$ and CNOT gates.
E. Smith et al.

Fig. 2. Example evolution of the search algorithm for a three-qubit circuit. It starts by placing a layer of single-qubit gates, then generating 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. 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.

judicious allocation of optimization time budget may improve scalability. Our multistart approach is discussed further in Section 6.

3 Prefix Circuit Synthesis

The synthesis solution space can be thought of as a tree that enumerates circuit structures of increasing depth: level 1 contains depth 1 structures, level 2 contains depth 2 structures, and so on. For scalability, we want to reach a solution while evaluating the least number of candidates possible and the shallowest circuits possible. The number of evaluations is given by the search algorithm: in the case of QSearch the path is driven by A*, and scalability is limited by long backtracking chains.

Our idea introduces a simple heuristic to reduce the frequency of backtracking. The approach is “data driven” and inspired by techniques employed in numerical optimization, as shown in Figure 3. Imagine mapping the search tree onto an optimization surface, which will contain plateaus and local minima. Exiting a plateau is characterized by faster progress toward a solution and minima. If the minima are local (partial solution is not acceptable), the algorithm has to walk out of the “valley.” Once out, the algorithm may still be on a plateau, but it can mark the region just explored as not “interesting” for any backtracking. The effect of implementing these principles in the search is illustrated in Figure 4. The result is a partitioning of the solution space into coarse-grained regions grouped by circuit depth range. During search, backtracking between solutions within a region is performed by using the A* rules. We never backtrack outside of a region to any candidate solution that resides in the previous “depth band.”
Overall, the effect of our strategy can be thought of as determining a prefix structure on the resulting circuit, as shown in Figure 4. The algorithm starts with a pure A* search on circuits up to depth \( d_1 \). The first depth \( d_1 \) viable partial solution is recorded, and the search proceeds to depth \( d_2 \) in subtree A. A* search proceeds in subtree A until finding the first viable candidate at depth \( d_2 \), then proceeds in subtree B. At this point, we have three regions: the start subtree for depth 0 to \( d_1 \), A for depth \( d_1 + 1 \) to \( d_2 \), and B for depth \( d_2 + 1 \) to \( d_3 \). In this example, the search in subtree B fails at depth \( d_2 + 1 \). We therefore backtrack to \( d_2 \), and the search proceeds on the path depicted on the right-hand side of the tree and eventually finds a solution.

One can easily see how by prohibiting backtracking into large solution subtrees we can reduce the number of evaluated (numerically optimized) candidates and improve scalability. As this changes the A* optimality property of the algorithm, the challenge is determining these subtrees in a manner that still leads to a short-depth solution.

Prefix formation. A partial solution describes a circuit structure and its function (gates). We have considered both static and dynamic methods for prefix formation. In our nomenclature, a static approach will choose a prefix circuit whose structure and function are fixed: this is a fully instantiated circuit. A dynamic approach will choose a fixed structure whose function is still parameterized. In the first case, the prefix circuit is completely instantiated with native gates to perform a single computation, whereas in the latter, it can “walk” a much larger Hilbert subspace as induced by the parameterization. Intuitively, determining a single instantiated prefix circuit is good for scalability. This reduces the number of parameters evaluated in any numerical optimization operation after
prefix formation. We have experimented with several strategies for forming instantiated prefix circuits in our synthesis algorithms, but they did not converge or they produced very long circuits.

Prefix formulation. In LEAP, we use a dynamic data-driven approach informed by the evolution of the underlying A* QSearch algorithm, described in Figure 4. Our analysis of the trajectories for multiple examples shows that many paths are characterized by a rapid improvement in solution quality (reduction in Hilbert–Schmidt distance between target unitary and approximate prefix), followed by plateauing induced either by optimizer limitations (local minima) or as an artifact of the particular structures considered (dead-end).

LEAP forms subtrees by first identifying and monitoring plateaus. Since during a plateau the rate of solution quality change is “low,” a “prefix” is formed whenever a solution is evaluated with a jump in the rate of change. The plateau identification heuristic is augmented with a work-based heuristic: we wait to form a prefix until we sample enough partial solutions on a path. This serves several purposes: it gives us more samples in a subtree to gain some confidence that we have not skipped “the only few viable partial solutions,” and it increases the backtracking granularity by identifying larger subtrees. Even more subtly, the work heuristic decreases the sensitivity of the approach to the thresholds used to assess the rate of change in the plateau identification method. By delaying to form a prefix based on work, we avoid jumping directly into another plateau that will result in superfluously evaluating many solutions that are close in depth to each other.

Solution optimality. By discarding pure A* search, LEAP gives up on always finding the optimal solution. However, the following observations based on the properties of the solution search space indicate that optimality loss could be small and that the approach can be generalized to other search and numerical optimization based methods. First, the solution tree of circuit structures exhibits high symmetry. Partial solutions can be made equivalent by qubit relabeling; all solutions reached from any equivalent structure will have a similar depth. For example, for a circuit with N qubits, a depth 1 circuit with a CNOT on qubits 0 and 1 can be thought of as “equivalent” to the circuit with a CNOT on qubits N − 2 and N − 1. Symmetry indicates that coarse-grained pruning may be feasible, since a subtree may contain many “equivalent” partial solutions.

Second, assuming that the optimal solution has depth \( d \), there are many easy-to-find solutions at depth \( > d \). In Figure 3, assume that the solution node \( S \) at depth \( d \) is missed by our strategy. However, there are links solutions at \( d + 1 \), links \( d + 2 \), and so forth, trivially obtained by adding identity gates to \( S \). In other words, the solution density increases (probably quadratically) with circuit depth increase. If the search has a “decent” partial solution at depth \( d \), numerical optimization is likely to find the final solution at very close depth. Overall, the high-level heuristic goal is to get to optimal depth with a “good enough” partial solution. Our “good enough” criteria combine the Hilbert–Schmidt norm with a measure of work.

The pseudocode for the prefix formation algorithm in LEAP is presented in Figure 5.
Algorithm 1 Helper Functions
1: function s(n)
2: return \( s(n) = CNOT + U_3 \) for all possible CNOT positions
3:
4: function \( r(n, U) \)
5: return \( \min_D(U(n, X), U) \)
6:
7: function \( h(d) \)
8: return \( d + a \) \( a \) is a constant determined via experiment
9:
10: function \( \text{predict_score}(a, b, d_i) \)
11: return \( \text{Predicted CNOTs for depth } d_i \text{ based on points in } a, b \)

Algorithm 2 LEAP Prefix Formation
1: function \( \text{LEAP\_SYNTHESIZE}(U_{\text{target}}, e, \delta) \)
2: \( s_i \leftarrow \text{the best score of prefixes} \)
3: \( n_i \leftarrow \text{the prefix structure} \)
4: while \( s_i > e \) do
5: \( n_i, s_i \leftarrow \text{INNER\_SYNTHESIZE}(U_{\text{target}}, e, \delta) \)
6: return \( n_i, s_i \)
7:
8: function \( \text{INNER\_SYNTHESIZE}(U_{\text{target}}, e, \delta) \)
9: \( n \leftarrow \text{representation of } U_3 \text{ on each qubit} \)
10: \( a \leftarrow \text{best depth values of intermediate results} \)
11: \( b \leftarrow \text{best depth values of intermediate results} \)
12: \( \text{push } n \text{ onto queue with priority } h(d_{\text{best}}) + 0 \)
13: while queue is not empty do
14: \( n \leftarrow \text{pop from queue} \)
15: for all \( n_i, U(n) \) do
16: \( s_i \leftarrow r(n_i, U_{\text{target}}) \)
17: \( d_i \leftarrow \text{CNOT count of } n_i \)
18: \( s_p \leftarrow \text{predict_score}(a, b, d_i) \)
19: if \( s_i < e \) then
20: return \( n_i, s_i \)
21: if \( s_i < s_p \) then
22: return \( n_i, s_i \)
23: if \( d_i < \delta \) then
24: \( \text{push } n_i \text{ onto queue with priority } h(d_i) + \text{CNOT count of } n_i \)

Fig. 5. Prefix formation algorithm in LEAP, based on the algorithm in the work of Davis et al. [6].

between partitions. This circuit is lifted to a unitary, and the reoptimizer synthesizes it and replaces it into the original solution. The process continues iteratively until a stopping criterion is reached. This amounts to moving a sliding optimization window across the circuit.

The quality of the solution is determined by the choice of the size of the resynthesis window, the number of applications (circuit coverage) and stopping criteria, and the numerical optimizer.

In LEAP, we make several pragmatic choices. The size of the optimization window is selected to be long enough for reduction potential but overall short enough that it can be optimized fast enough. The algorithm reoptimizes exactly once at each boundary in the original partitioning. The resynthesis pass allows us to manage the budget given to numerical optimizers. Since each circuit piece is likely to be transformed multiple times, some of the operations can use fast but lower-quality/budget optimization. We do use the fastest optimizer available during prefix synthesis, switching during resynthesis to the higher-quality but slower multistart solver based on the work of Larson and Wild [13], described in Section 6.

5 DIMENSIONALITY REDUCTION

The circuit solution provides a parameterized structure instantiated for the solution. This parameterization introduced by the single-qubit \( U_3 \) gates may overfit the problem.

For LEAP, which targets only the CNOT count, this may be a valid concern, and we therefore designed a dimensionality reduction pass. We use a simple algorithm that attempts to delete one \( U_3 \) gate at a time and reinstantiates the circuit at each step. This linear complexity algorithm can discover and remove only simple correlations between parameters. More complex cases can be discovered borrowing from techniques for dimensionality reduction for machine learning [24] or numerical optimization [25].

When applied to the final synthesis solution, dimensionality reduction may reduce the circuit critical path even further by deleting \( U_3 \) gates. It can also also be combined with the prefix synthesis. Once a prefix is formed, we can reduce its dimensionality. As numerical optimizers scale
exponentially with parameters, this will improve the execution time per invocation. However, it may affect the quality of the solution as we remove expressive power from prefixes. In the current LEAP version, only the final solution is simplified.

6 MULTISTART OPTIMIZATION

Solving the optimization problem for the objective function in LEAP or QSearch can be difficult. Quantum circuits, even optimal ones, are not unique: a global phase is physically irrelevant and thus does not affect the output. Furthermore, circuits that differ only in a local basis transformation and its inverse surrounding a circuit subsection (e.g., a single two-qubit gate) are mathematically equivalent.\(^3\) Provided native gate sets may contain equivalences; single-qubit gates, being rotations, are periodic. As a practical matter, we find that we cannot declare these equivalences to existing optimizers. Furthermore, where they can be used to create constraints or inaccessible regions (e.g., by remapping the periodicity into a single region), we find that they hinder the search, because boundaries can create artificial local minima.

The unavoidable presence of equivalent circuits means that we are essentially overfitting the problem, where changes in parameters can cancel each other out, leading to saddle points, which turn into local minima in the optimization surface because of the periodicity; see Figure 6. The former cause, at best, an increase in the number of iterations as progress slows down because of smaller gradients; the latter risks getting the optimizer stuck.

Another problem comes from the specification of the objective: distance metrics care only about the output, and different circuits can thus result in equal distances from the desired unitary. If no derivatives are available, this results in costly evaluations just to determine that no progress can be made, a problem that gets worse at scale. But even with a derivative, it closes directions for exploration and shrinks viable step sizes, thus increasing the likelihood of getting stuck in a local minimum.

In sum, local optimization methods are highly dependent on the starting parameters, yet global optimization methods can require far too many evaluations to be feasible for real-world objectives. An attractive middle ground is an approach that starts many local optimization runs from different

---

\(^3\)There are physical differences; in particular, such circuits tend to sample different noise profiles. This property forms the basis of randomized compilation.

ACM Transactions on Quantum Computing, Vol. 4, No. 1, Article 5. Publication date: February 2023.
points in the domain. Multistart optimization methods are especially appealing when there is some structure in the objective, such as the least-squares form of the objective.

Some multistart approaches complete a given local optimization run before starting another, whereas others may interleave points from different runs. The asynchronously parallel optimization solver for finding multiple minima (APOSMM) \cite{13} begins with a uniform sampling of the domain and then starts local optimization runs from any point subject to constraints: (1) point not yet explored, (2) not a local optimum, and (3) no point available within a distance $r_k$ with a smaller function value. If no such point is available, more sampling is performed. The radius $r_k$ decreases as more points are sampled, thereby allowing past points to start runs. Under certain conditions on the objective function and the local optimization method, the logic of APOSMM can be shown to asymptotically identify all local optima while starting only finitely many local optimization runs.

7 EXPERIMENTAL SETUP

LEAP, available at https://github.com/BQSKit/qsearch, extends QSearch. We evaluated it with Python 3.8.5, using numpy 1.19.5 and Rust 1.48.0 code.

For our APOSMM implementation, we integrated with the version in the libEnsemble Python package \cite{26,27}. We tried two different local optimization methods within APOSMM: the L-BFGS implementation within SciPy \cite{22} and the Google Ceres \cite{23} least-squares optimization routine.

For experimental evaluation, we use a 3.35-GHz Epyc 7702p based server, with 64 cores and 128 threads. Our workload consists of known circuits (e.g., mul, add, Quantum Fourier Transform), as well as newly introduced algorithms. VQE \cite{15} starts with a parameterized circuit and implements a hybrid algorithm where parameters are reinstantiated based on the results of the previous run. The TFIM \cite{16} and QITE \cite{18} algorithms model the time evolution of a system. They are particularly challenging for NISQ devices as circuit length grows linearly with the simulated timestep. In TFIM, each timestep (extension) can be computed and compiled ahead of time from first principles, whereas in QITE, it is dependent on the previous timestep.

We evaluate LEAP against QSearch and other available state-of-the-art synthesis software and compilers. QFAST \cite{28} scales better than QSearch by conflating search for structure with numerical optimization, albeit producing longer circuits. Qiskit-synth \cite{10} uses linear algebra decomposition rules for fast synthesis, but circuits tend to be long. IBM Qiskit \cite{8} provides “traditional” quantum compilation infrastructures using peephole optimization and mapping algorithms. CQC Tket \cite{9} proves another good quality compilation infrastructure across multiple gate sets. To showcase the impact of QPU topology, we compile for processors where qubits are fully connected (all-to-all), as well as processors with qubits connected in a nearest-neighbor (linear) fashion.

8 EVALUATION

Summarized results are presented in Table 2, with more details in Tables 1 and 4. We present data for all-to-all and nearest-neighbor chip topology.

Table 3 presents a direct comparison between QSearch and LEAP for circuits up to four qubits. Despite its heuristics, LEAP produces optimal-depth solutions, matching the reference implementations on nearest-neighbor chip topology. Overall, LEAP can compile four-qubit unitaries up to 59× faster than QSearch.

As shown in Table 4, LEAP scales up to six qubits. In this case, we include full topology data, as well results for compilation with QFAST, Qiskit, Qiskit-synth, and Tket. On well-known quantum circuits such as VQE and QFT and physical simulation circuits such as TFIM, LEAP with resynthesis can reduce the CNOT count by up to 48×, or 11× on average when compared to Qiskit. On average when compared to Tket, LEAP reduces depth by a factor of 7×. Our heuristics rarely affect solution quality, and LEAP can match optimal-depth solutions. At five and six qubits, LEAP synthesizes
circuits with to 1.19× fewer CNOTs on average compared with QFAST, albeit with an average 3.55× performance penalty. LEAP can be one order of magnitude slower than Qiskit-synth while providing two or more orders of magnitude shorter circuits.

### 8.1 Impact of Prefix Synthesis

Most of the speed improvements are directly attributable to prefix synthesis, which reduces by orders of magnitude the number of partial solutions evaluated. For example, for QFT4, the whole search space contains ≈43M solution candidates. QSearch will explore 2,823 circuits, whereas LEAP will explore 410. For TFIM-22, these numbers are (≈1.6M, 54,020, 176), respectively. Detailed results are omitted for brevity.

### 5.12 E. Smith et al.

| Depth | Qiskit Synthesized | TKet Mapped | LEAP | QFAST | Qiskit Mapped | Qiskit Synthesis |
|-------|--------------------|-------------|------|-------|---------------|------------------|
| 3     | 221               | 215.47      | 28   | 48.6  | 9,004         | ≈ 291            |
| 4     | 321               | 270.27      | 42.7 | 78.2  | 9,512         | 2.9e-1*          |
| 5     | 621               | 4,169       | 785  | 851   | 16,410        | 4,169            |
| 6     | 1,062             | 227         | 241  | 292   | 423           | 1,062            |
| 7     | 1,423             | 421         | 421  | 421   | 423           | 1,423            |

*Three-qubit results were chosen as the best run of two samples.

**Table 2. Summary of the Quality Metrics (Average Value) for Five- and Six-Qubit Circuit Synthesis**

| All-to-All | Qiskit Mapped | TKet Mapped | LEAP | QFAST | Qiskit Mapped | Qiskit Synthesis |
|------------|---------------|-------------|------|-------|---------------|------------------|
| Time (s)   | 0.06          | 0.20        | 0.44 | 0.85  | 1.52          | 1.58             |
| Error      | 1e-16         | 1e-15       | 1.62 | 2.07  | 1.95          | 2.07             |
| CNOT       | 240           | 240         | 285  | 285   | 285           | 285              |
| U_1         | 270           | 243.07      | 41.71| 60.9  | 2,155         | 291              |
| Depth      | 207           | 206.67      | 29.2 | 43.9  | 3,912         | 321              |

*Qiskit’s methods are exact, yet due to some post-processing in their mapping pipeline, large errors are shown.
Table 3. Summary of Synthesis Results for QSearch and LEAP on the Linear Topology

| QSearch | LEAP |
|---------|------|
| ALG     | Qubits | Ref | CNOT | Unitary Distance | Time(s) | CNOT | Unitary Distance | Time(s) |
| QFT     | 3      | 6   | 7    | 5.33 + 10^{-6} | 2.0    | 8    | 2.22 + 10^{-6} | 1.7    |
| Toffoli | 3      | 8   | 3    | 8.22 + 10^{-6} | 3.4    | 8    | 2.22 + 10^{-6} | 1.6    |
| Fredkin | 3      | 8   | 8    | 4.44 + 10^{-6} | 2.6    | 8    | 3.33 + 10^{-6} | 1.7    |
| Peres   | 3      | 7   | 7    | 1.72          | 1.2    | 7    | 2.22 + 10^{-6} | 1.1    |
| Logical OR | 3  | 6   | 6    | 2.22 + 10^{-6} | 3.4    | 8    | 3.33 + 10^{-6} | 1.6    |
| QFT     | 4      | 12  | 14   | 6.7 + 10^{-6} | 2.429  | 13   | 6.7 + 10^{-6} | 77.9   |
| TFIM-1  | 4      | 6   | 6    | 0            | 13.4   | 6    | 0             | 7.2    |
| TFIM-10 | 4      | 60  | 11   | 9.08 + 10^{-11} | 955.4  | 11   | 3.95 + 10^{-11} | 47.8   |
| TFIM-22 | 4      | 126 | 12   | 1.22 + 10^{-5} | 2.450  | 12   | 3.77 + 10^{-5} | 41.6   |
| TFIM-60 | 4      | 360 | 12   | 4.44 + 10^{-6} | 1.391  | 12   | 2.22 + 10^{-6} | 31.6   |
| TFIM-80 | 4      | 480 | 12   | 4.44 + 10^{-6} | 1.553  | 12   | 2.22 + 10^{-6} | 35     |
| TFIM-95 | 4      | 570 | 12   | 6.66 + 10^{-6} | 1.221  | 14   | 2.22 + 10^{-6} | 38.1   |

LEAP produces very similar results as QSearch in significantly less time.

Table 4. Results for Five- and Six-Qubit Synthesis Benchmarks with QFAST, LEAP, and IBM Qiskit

| CNOTs | All-to-All | Linear |
|-------|------------|--------|
| Qiskit Mapped | Qiskit Mapped | Qiskit Mapped |
| Quiskt Mapped | Qiskit Mapped | Qiskit Mapped |
| Depth | U(s) |
|-------|------|
| Five Qubits | Six Qubits |
| GFNS | hlf | mil | qua | qft | QFAST-10 | QFAST-40 | QFAST-60 | QFAST-80 | QFAST-100 | QFAST-10-1 | QFAST-24 | QFAST-31 | QFAST-51 | QFAST-10-2 | QFAST-24 | QFAST-31 | QFAST-51 |
| Quiskt Mapped | Quiskt Mapped | Quiskt Mapped |
| Quiskt Mapped | Quiskt Mapped | Quiskt Mapped |
| LEAP | LEAP | LEAP | LEAP | LEAP | LEAP | LEAP |
| QFAST | QFAST | QFAST | QFAST | QFAST | QFAST | QFAST |
| Peres | Peres | Peres | Peres | Peres | Peres | Peres |
| QFAST | QFAST | QFAST | QFAST | QFAST | QFAST | QFAST |
| Depth | U(s) |
|-------|------|
| Five Qubits | Six Qubits |
| GFNS | hlf | mil | qua | qft | QFAST-10 | QFAST-40 | QFAST-60 | QFAST-80 | QFAST-100 | QFAST-10-1 | QFAST-24 | QFAST-31 | QFAST-51 | QFAST-10-2 | QFAST-24 | QFAST-31 | QFAST-51 |
| Quiskt Mapped | Quiskt Mapped | Quiskt Mapped |
| Quiskt Mapped | Quiskt Mapped | Quiskt Mapped |
| LEAP | LEAP | LEAP | LEAP | LEAP | LEAP | LEAP |
| QFAST | QFAST | QFAST | QFAST | QFAST | QFAST | QFAST |
| Peres | Peres | Peres | Peres | Peres | Peres | Peres |
| QFAST | QFAST | QFAST | QFAST | QFAST | QFAST | QFAST |
| Depth | U(s) |
|-------|------|
| Five Qubits | Six Qubits |
| GFNS | hlf | mil | qua | qft | QFAST-10 | QFAST-40 | QFAST-60 | QFAST-80 | QFAST-100 | QFAST-10-1 | QFAST-24 | QFAST-31 | QFAST-51 | QFAST-10-2 | QFAST-24 | QFAST-31 | QFAST-51 |
| Quiskt Mapped | Quiskt Mapped | Quiskt Mapped |
| Quiskt Mapped | Quiskt Mapped | Quiskt Mapped |
| LEAP | LEAP | LEAP | LEAP | LEAP | LEAP | LEAP |
| QFAST | QFAST | QFAST | QFAST | QFAST | QFAST | QFAST |
| Peres | Peres | Peres | Peres | Peres | Peres | Peres |
| QFAST | QFAST | QFAST | QFAST | QFAST | QFAST | QFAST |

Note: The asterisk (*) implies that the program timed out after 12 hours.

Prefix formation is calculated based on a best-fit line formed by a linear regression of the best scores versus the depth associated with the new best-found score. This linear regression is used as an estimator of the expected score at the current depth. When the score calculated from the heuristic is better than the expected score, this means that the new best score is better than expected—in other words, the expected progress to the solution is slower, and the calculated score is worse than the expected score. We therefore do not form prefixes in this case, which allows LEAP to maintain the important backtracking and searching that makes QSearch optimal.

Prefix formation is calculated based on a best-fit line formed by a linear regression of the best scores versus the depth associated with the new best-found score. This linear regression is used as an estimator of the expected score at the current depth. When the score calculated from the heuristic is better than the expected score, this means that the new best score is better than expected—in other words, the expected progress to the solution is slower, and the calculated score is worse than the expected score. We therefore do not form prefixes in this case, which allows LEAP to maintain the important backtracking and searching that makes QSearch optimal.

ACM Transactions on Quantum Computing, Vol. 4, No. 1, Article 5. Publication date: February 2023.
Table 5. Number and Location of Prefix Blocks for Various Circuits

| ALG    | Qubits | CNOT  | # of Blocks | Block End Locations |
|--------|--------|-------|-------------|---------------------|
| fredkin | 3      | 8     | 2           | 5.8                 |
| toffoli | 3      | 8     | 2           | 6.9                 |
| grover3 | 3      | 7     | 2           | 5.7                 |
| bhl     | 3      | 7     | 1           | 3                   |
| or      | 3      | 8     | 2           | 5.8                 |
| peres   | 3      | 7     | 2           | 6.7                 |
| qft3    | 3      | 8     | 2           | 5.9                 |
| qft4    | 4      | 18    | 4           | 5.13,18,21          |
| adder   | 4      | 15    | 3           | 8.14,19             |
| vqe     | 5      | 20    | 8           | 3,7,11,14,18,21,25,28 |
| TFIM-1  | 4      | 7     | 2           | 5.7                 |
| TFIM-10 | 4      | 12    | 3           | 5.10,12             |
| TFIM-22 | 4      | 12    | 3           | 5.10,12             |
| TFIM-60 | 4      | 12    | 3           | 5.10,12             |
| TFIM-80 | 4      | 12    | 3           | 5.10,12             |
| TFIM-95 | 4      | 12    | 3           | 5.10,12             |
| mul     | 5      | 15    | 5           | 3.9,12,16,18        |
| qaoa    | 5      | 28    | 7           | 6,10,14,19,24,29,35 |
| qft5    | 5      | 30    | 11          | 5,8,11,15,20,23,30,35,38,40 |
| TFIM-10 | 5      | 18    | 7           | 3.6,9,13,16,19,21   |
| TFIM-40 | 5      | 20    | 7           | 3.7,10,13,16,19,21  |
| TFIM-60 | 5      | 20    | 7           | 3.6,10,15,18,21,24  |
| TFIM-80 | 5      | 20    | 7           | 3.6,11,16,20,23,24  |
| TFIM-100| 5      | 20    | 6           | 5.9,13,17,20,22     |
| TFIM-1  | 6      | 10    | 4           | 4.7,10,12           |

Table 6. Summary of the CNOT Reduction and Time for Resynthesis on the Linear Topology

| ALG    | Qubits | CNOT Before Resynthesis | Unitary Distance | Time (s) | CNOT After Resynthesis | Unitary Distance | Time (s) |
|--------|--------|--------------------------|------------------|----------|------------------------|------------------|----------|
| qft3   | 3      | 8.44 * 10^{-16}          | 1.4              | 10.8     | 8.44 * 10^{-16}        | 1.4              | 5.9      |
| logical or | 3     | 2.22 * 10^{-16}          | 1.4              | 10.8     | 2.22 * 10^{-16}        | 1.4              | 5.7      |
| fredkin| 3      | 2.22 * 10^{-16}          | 1.7              | 10.8     | 0                      | 3.4              | 3.4      |
| toffoli | 3      | 2.22 * 10^{-16}          | 1.7              | 10.8     | 0                      | 3.4              | 3.4      |
| adder  | 4      | 8.03 * 10^{-12}          | 10.3             | 10.8     | 8.03 * 10^{-12}        | 17.66            | 17.66    |
| qft4   | 4      | 6.66 * 10^{-16}          | 4.2              | 10.8     | 6.66 * 10^{-16}        | 17.66            | 17.66    |
| TFIM-10| 4      | 8.03 * 10^{-12}          | 10.3             | 10.8     | 8.03 * 10^{-12}        | 17.66            | 17.66    |
| TFIM-95| 4      | 4.44 * 10^{-16}          | 6.5              | 10.8     | 4.44 * 10^{-16}        | 113              | 113      |
| vqe    | 4      | 2.47 * 10^{-11}          | 151.1            | 10.8     | 2.70 * 10^{-11}        | 2.062             | 2.062    |
| qft5   | 5      | 1.22 * 10^{-15}          | 772.4            | 10.8     | 6.66 * 10^{-16}        | 4.392             | 4.392    |
| TFIM-10| 5      | 7.97 * 10^{-12}          | 310.6            | 10.8     | 9.19 * 10^{-12}        | 11,320.8         | 11,320.8 |
| TFIM-40| 5      | 6.66 * 10^{-16}          | 44               | 10.8     | 0                      | 3.5418           | 3.5418   |
| TFIM-60| 5      | 6.66 * 10^{-16}          | 66.9             | 10.8     | 0                      | 2.0465           | 2.0465   |
| TFIM-40| 5      | 2.22 * 10^{-16}          | 73.5             | 10.8     | 0                      | 1.287             | 1.287    |
| TFIM-100| 5     | 4.44 * 10^{-16}          | 55.4             | 10.8     | 1.11 * 10^{-16}        | 2.779             | 2.779    |
| mul    | 5      | 4.44 * 10^{-16}          | 47.0             | 10.8     | 1.11 * 10^{-16}        | 809.2             | 809.2    |
| TFIM-1 | 6      | 2.22 * 10^{-16}          | 213.3            | 10.8     | 1.11 * 10^{-16}        | 7,437.9          | 7,437.9  |

Table 5 presents the number of prefixes formed during synthesis for each circuit considered. Since prefixes have a depth between three and five qubits, this informs our choice of the resynthesis window discussed in the following.

8.2 Impact of Incremental Resynthesis

While significantly reducing depth (with respect to the circuit reference), prefix synthesis can be improved upon by incremental resynthesis, as shown by the comparison in Table 6. LEAP applies only a single step of resynthesis. Given the solution from prefix synthesis, LEAP selects a window at each prefix boundary, resynthesizes, and reassembles the circuit. Detailed results are omitted for brevity, but further iterations do little to improve the solution.
### Table 7. Spatial Placement of $U_3$ Gates Deleted

| Name  | Number of Gates Deleted |
|-------|--------------------------|
| qft2  | 2 0 0 0 0 0             |
| qft3  | 2 0 0 1 0 1 1 0 1      |
| fredkin | 3 2 0 1 1 2 0 0 1 0 0 1 0 0 1 0 1 0 |
| toffoli | 2 2 1 2 1 2 0 1 0 1 0 1 0 1 0 1 0 1 0 |
| peres  | 2 0 1 2 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 |
| logical_or | 2 1 2 0 2 1 0 1 0 1 0 1 0 1 0 1 0 1 0 |

**Note:** The number of columns denotes circuit stages (CNOTs), and we present the number of gates deleted at each position.

### Table 8. Accuracy and Speed of Various Optimizers on a Variety of Circuits

| ALG | CNOT | BFGS | Ceres | APOSMM-8 | APOSMM-12 | APOSMM-16 | APOSMM-20 | APOSMM-24 |
|-----|------|------|-------|----------|-----------|-----------|-----------|-----------|
|     | % Success | Time (s) | % Success | Time (s) | % Success | Time (s) | % Success | Time (s) | % Success | Time (s) | % Success | Time (s) | % Success | Time (s) | % Success | Time (s) | % Success | Time (s) |
| fredkin | 8 | 16 | <0.01 | 75 | 0.01 | 100 | 0.13 | 100 | 0.14 | 100 | 0.15 | 100 | 0.16 | 100 | 0.17 | 100 | 0.18 |
| peres | 7 | 15 | <0.01 | 75 | 0.01 | 100 | 0.13 | 100 | 0.14 | 100 | 0.15 | 100 | 0.16 | 95 | 0.13 | 95 | 0.14 |
| toffoli | 8 | 43 | 0.01 | 74 | 0.01 | 100 | 0.13 | 100 | 0.14 | 100 | 0.15 | 100 | 0.16 | 100 | 0.17 | 100 | 0.18 |
| gft2 | 8 | 0 | <0.01 | 25 | 0.01 | 80 | 0.03 | 97 | 0.12 | 99 | 0.13 | 98 | 0.14 | 98 | 0.15 | 99 | 0.16 |
| gft3 | 13 | 0 | <0.01 | 66 | 0.03 | 83 | 0.10 | 92 | 0.19 | 94 | 0.20 | 99 | 0.21 | 99 | 0.22 | 99 | 0.23 |
| gft5 | 90 | 0 | <0.01 | 2 | 0.12 | 8 | 1.19 | 15 | 2.78 | 15 | 3.81 | 25 | 7.21 | 36 | 12.10 |

**Note:** APOSMM-N means APOSMM with N starting points.

The resynthesis window in LEAP is chosen pragmatically with a limited depth (seven CNOTs for three and four qubits, five CNOTs for five and six qubits in our case), to lead to reasonable expectations on execution time, while providing some optimization potential.

Incremental resynthesis reduces circuit depth by 15% on average, albeit in many cases with a significant impact on the runtime.

#### 8.3 Impact of Dimensionality Reduction

LEAP applies a single step of dimensionality reduction at the end of the synthesis process, the sweep starting at the circuit beginning. For brevity, we omit detailed data and note that in this final stage dimensionality reduction eliminates up to 40% of $U_3$ gates (parameters) and shortens the circuit critical path. These results indicate that our approach overfits the problem by inserting too many $U_3$ gates.

We examined the spatial occurrence of single-qubit gate deletion since this may guide any dynamic attempts to eliminate parameters during synthesis for scalability purposes. Table 7 presents a summary for three-qubit circuits; trends are similar for all other benchmarks considered. The data shows that gate deletion is successful at many circuit layers, indicating that a heuristic for on-the-fly dimensionality reduction heuristic may be feasible to develop for even further scalability and quality improvements. As discussed in Section 6, dimensionality reduction will reduce the number of parameters for numerical optimization while reducing overfitting and gate (parameter) correlation that lead to cancellations of gate effects on a qubit.

#### 8.4 Impact of Multistart Optimization

When evaluating numerical optimizers used in synthesis, we are interested in determining how often they found the true minimum, as this has a significant impact on both solution quality and execution speed. We evaluated the commonly used local optimization methods Google’s Ceres [23] and an implementation of L-BFGS [22] as well as the multistart APOSMM [13] framework.

We ran each optimizer 100 times on several circuits to evaluate their accuracy and speed. The results are summarized in Table 8. The QFT results illustrate that the BFGS and Ceres optimizers perform poorly even on a smaller circuit such as a three-qubit QFT, finding solutions just 9% and 26% of the time, much lower than even APOSMM with 8 starting points. We found that APOSMM
with 12 starting points performed well on all but the five-qubit QFT circuit. Since optimizing the parameters of the QFT5 circuit is a much higher-dimensional problem, even APOSMM with 24 starting points found solutions in only 36% of the runs.

While APOSMM is much more accurate than BFGS and Ceres on the circuits we tested, it is also about an order of magnitude slower for larger circuits, even though the local optimization runs are done in parallel. In addition, the slowdown increases with the number of starting points. The time for QFT5 approximately doubles every 4 additional starting points for parallel runs. For our runs in Table 4, we selected 12 starting points since this number was reasonably accurate and takes a reasonable amount of time.

Therefore when using LEAP, we use Ceres because it is fast and scales well, and a missed solution will be found during resynthesis. During resynthesis, APOSMM is used, as it is much more likely to find true minima, thus strengthening the optimality of search-based algorithms.

8.5 Gate Set Exploration

Similar to QSearch, LEAP can target different native gate sets and provide another dimension to circuit optimization or hardware design exploration. Besides CNOT, we have targeted other two-qubit gates supported by QPU manufacturers: CSX (\(\sqrt{\text{CNOT}}\)), iSWAP, and SQISW (\(\sqrt{iSWAP}\)). Here, the square root gates implement the matrix square root of their counterpart, and their composition has been previously studied\[29\] for generic two-qubit programs. Results are presented in Table 9. We make the following observations:

- Although CNOT and iSWAP are considered “equivalent” in terms of expressive power, using CNOT gates for larger circuits (five and six qubits) tends to produce observably shorter circuits.
- Mixing two-qubit gates (CNOT+iSWAP) tends to produce shorter circuits than when using CNOT alone.
- The depths of CNOT- and \(\sqrt{\text{CNOT}}\)-based circuits are very similar. Given that in some implementations the latency of \(\sqrt{\text{CNOT}}\) gates may be shorter than that of CNOT gates, the former may be able to provide a performance advantage.
- Sleator and Weinfurter\[30\] prove that the Toffoli gate can be optimally implemented using a five-gate combination of CNOT and \(\sqrt{\text{CNOT}}\). LEAP can reproduce this result, which indicates that it may provide a useful tool for discovering optimal implementations of previously proposed gates.

These observations are somewhat surprising and probably worth a more detailed future investigation. The data indicates that mixing CNOT and iSWAP can produce the shortest circuits; however, we found that in LEAP the search space size would double, hence the speed to the solution will suffer. Therefore, for our experiments, we kept with the CNOT+U3 gate set that was used by QFAST and QSearch.

9 DISCUSSION

Overall, the results indicate that the heuristics employed in LEAP are much faster than QSearch and are still able to produce low-depth solutions in a topology-aware manner. The average depth difference for three- and four-qubit benchmarks between QSearch and LEAP is 0 across physical chip topologies and workload.

We find the prefix formation idea intuitive, easily generalizable, and powerful. The method used to derive prefix formation employs concepts encountered in numerical optimization algorithms
Table 9: Number of Two-Qubit Gates Needed to Implement Various Three- to Six-Qubit Circuits

| ALG     | CNOT | SQCNOT | iSWAP | SQISW | CNOT + iSWAP | CNOT + SQCNOT | iSWAP + SQISW |
|---------|------|--------|-------|-------|--------------|---------------|---------------|
| qft3    | 6    | 8      | 7     | 8     | 5            | 5             | 7             |
| fredkin | 7    | 9      | 7     | 9     | 7            | 7             | 8             |
| toffoli | 6    | 7      | 7     | 8     | 6            | 5             | 7             |
| peres   | 5    | 5      | 7     | 8     | 5            | 4             | 6             |
| logical or | 6   | 7      | 7     | 8     | 6            | 8             | 7             |

| ALG     | iSWAP | CNOT |
|---------|-------|------|
| qft4    | 22    | 13   |
| tfim-4-22 | 16  | 12   |
| tfim-4-95 | 14  | 12   |
| vqe     | 26    | 21   |
| full adder | 30  | 18   |
| hlf     | 22    | 13   |
| mul     | 18    | 13   |
| qft5    | 50    | 28   |
| tfim-5-40 | 29  | 20   |
| tfim-5-100 | 33 | 20   |
| tfim-6-24 | 40  | 28   |
| tfim-6-51 | 43  | 31   |

Note: Using CNOT reduces the number of two-qubit gates needed vs iSWAP, whereas a combination of CNOT and iSWAP reduces the number of two-qubit gates even further.

Fig. 7. TFIM circuit depth evolution and “fidelity” when executed on the IBM Athens system. “IBM” is compiled with Qiskit, whereas “constant depth” is synthesized with LEAP.

and is easily identifiable in other search-based synthesis algorithms: “progress” to the solution, and “region of similarity” or plateau.

The LEAP algorithm indicates that incremental and iterative approaches to synthesis work well. In our case, the results even indicate that one extra step of local optimization can match the efficacy of global optimization. This result bodes well for approaches that scale synthesis past hundreds of qubits through circuit partitioning, such as our QGo [31] optimization and QuEst [32] approximation algorithms.

Dimensionality reduction as implemented in LEAP not only reduces the effects of overfitting by numerical optimization but also opens a promising path for scaling numerical optimization based synthesis. Since we were able to delete 40% of parameters from the final solution, we believe that by combining it with prefix synthesis we can further improve LEAP’s scalability.

Multistart optimization can be trivially incorporated into any algorithm, and we have indeed already modified the QSearch and QFAST algorithms to incorporate it. Furthermore, the spirit of the multistart “approach” can be employed to further prune the synthesis search space. Whenever a prefix formed, the synthesis algorithm had explored a plateau and a local minimum. At this stage, a multistart search could be started using as seeds other promising partial solutions within the tree.

9.1 Scaling Hamiltonian Simulation Circuit Exploration: TFIM and QITE

The prefix formation idea is powerful and showcases how synthesis can turn into a capability tool. TFIM circuits simulate a time-dependent Hamiltonian, where the circuit for each timestep
“contains” the circuit (computation) associated with the previous time step as a prefix. The circuits generated by the TFIM domain generator grow linearly in size. In our experiments, we observed that after some initial timesteps, all circuits for any late timestep have an asymptotic constant depth. This observation led to the following experiment: we picked a circuit structure generated for a late simulation step and considered it as a parameterized template for all other simulation steps. We then successfully solved the numerical optimization problem with this template for any TFIM step. This procedure empirically provides us with a fixed-depth (short-depth) template for the TFIM algorithm. Furthermore, this demonstration motivated a successful effort [33] to derive from first principles a fixed-depth circuit for TFIM. The results are presented in Figure 7. Note the highly increased fidelity when running the circuit on the IBM Athens system.

The QITE algorithm presents an interesting challenge to the prefix formation idea. In this case, the next timestep circuit is obtained by extending the “current” circuit with a block dependent on its output after execution. When executing on hardware, synthesis has real-time constraints, and it has to deal with the hardware noise that affects the output. Preliminary results, courtesy of our collaborators Jean-Loup Ville and Alexis Morvan, indicate that the approach taken for TFIM may be successful for QITE. Table 10 summarizes the preliminary observations and indicates that again synthesis produces better-quality circuits than the domain generator or traditional compilation does. Note that in this experiment, LEAP was fast enough to produce real-time results during the hardware experiment only for three-qubit circuits.

### 9.2 Analysis of Runtime Scaling as a Synthesis Application

Like other synthesis algorithms (that operate on the complete unitary as input), LEAP scales exponentially with the number of qubits. To compare the runtime to other synthesis algorithms, we consider the effective branching factor of the search in LEAP.

To evaluate the complexity of LEAP’s algorithm, we calculated the effective branching factor of the search. The effective branching factor equation, from Russell and Norvig [34], is given by

\[ N + 1 = 1 + b^* + (b^*)^2 + \cdots + (b^*)^d, \]  

where \( d \) is the depth of the search. We evaluated the effective branching factor of LEAP’s search on multiple circuits, presented in Figure 8. The entries are sorted by qubit count, then by circuit length. As can be seen, there appears to be a negative correlation between final depth of the solution and effective branching factor, indicating that LEAP’s pruning is effective at shrinking the search space. The best effective branching factor evaluated was TFIM-5-80, with \( b^* = 1.08 \). The worst effective branching factor was QFT 3, with \( b^* = 1.41 \). The average effective branching factor was \( b^* = 1.22 \).

Therefore, the overall runtime complexity of LEAP is approximately \( O(2^{3n} \cdot d \cdot b^*^d) \), which we can simplify to \( O(2^{3n+\log_2(d)}+\log_2(b^*)) \) by grouping to a common base. Since \( \log_2(d) \leq \log_2(b^*) \), we can further simplify to the simplified complexity of \( O(2^{3n+\log_2(b^*)}) \), where \( n \) is the number of qubits, \( b^* = 1.22 \) the average effective branching factor, and \( d = \lceil \frac{1}{4}(4^n - 3n - 1) \rceil \) is the minimum number

of layers built by CNOTs plus one qubit gates to produce an arbitrary $n$-qubit gate according to Shende et al. [35]. Note that when possible, LEAP generally finds a shorter solution when possible, so the complexity given is an upper bound on what we expect from LEAP’s performance.

UniversalQ, another synthesis algorithm, has runtime complexity $O(2^{3n})$ [10] for decomposing isometries, which is dominated by matrix operations similar to what LEAP requires for simulation. However, LEAP also must search over permutations, which adds the extra factor to the complexity.

### 9.3 Application to General Circuit Optimization

LEAP, like Qsearch, synthesizes the whole unitary. Thus, the complexity of the algorithm is exponential in the number of qubits. However, as seen in the past two examples on Hamiltonian simulation, LEAP has already seen use in synthesizing smaller circuits where the impact of the exponential scaling is not felt.

For synthesizing larger circuits, LEAP can be an important building block for partitioning-based circuit optimizers. Partitioning-based circuit optimization runtime scales at the rate of synthesis of each block, and linearly with the number of blocks. Therefore, if the size of the block is kept small, the dominant scaling is the number of blocks to synthesize, which can be done in parallel.

For such partitioned circuits, high-quality block optimization is vital to the quality of the overall optimization. Therefore, LEAP provides a compelling block synthesizer for partitioners, as it provides high-quality results for smaller block sizes.

Looking further forward, the question remains whether numerical optimization based synthesis can be useful in fault-tolerant quantum computing. There, the single-qubit gates will change to Cliffords and the T gate, or another non-Clifford gate that makes the gate set universal. The execution cost model is also expected to be different: CNOTs and Cliffords become cheap, whereas the non-Clifford operations become expensive. Likely, the non-Cliffords are qualitatively more “expensive” than CNOTs in NISQ computing. Thus, the optimization objective becomes minimizing the number of non-Clifford gates.

We have already shown that LEAP can be retargeted to new gate sets. We also have very strong evidence that adding a multiobjective optimization approach to search-based synthesis works very well under a fault-tolerant quantum computing cost model. The data indicates that it is realistic to
expect efficacy improvements similar to those provided by LEAP under the NISQ cost model. This work is ongoing (and due to intellectual property concerns, we cannot disclose more details). As the already mentioned scalable partitioning approaches only leverage LEAP and do not require additional cost models, this bodes very well for the future of numerical optimization based synthesis in fault-tolerant quantum computing.

10 RELATED WORK

A fundamental result that spurred the apparition of quantum circuit synthesis is provided by the Solovay–Kitaev theorem. The theorem relates circuit depth to the quality of the approximation, and its proof is by construction [36–38]. Different approaches [36, 39–48] to synthesis have been introduced since, with the goal of generating shorter-depth circuits. These can be coarsely classified based on several criteria: target gate set, algorithmic approach, and solution distinguishability.

**Target gate set.** The Solovay–Kitaev algorithm is applicable to any universal gate set. Later examples include synthesis of z-rotation unitaries with Clifford+V approximation [49] or Clifford+T gates [50]. When ancillary qubits are allowed, one can synthesize single-qubit unitaries with the Clifford+T gate set [50–52]. Although these efforts propelled the field of synthesis, they are not used on NISQ devices, which offer a different gate set (Rx, Rz, CNOT, iSWAP and Mølmer–Sørensen all-to-all). Several [1–3] other algorithms, discussed in the following, have since emerged.

**Algorithmic approaches.** The early attempts inspired by the Solovay–Kitaev algorithm 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 [41] algorithm.

Several approaches use techniques from linear algebra for unitary and tensor decomposition. Bullock and Markov [44] use QR matrix factorization via a Givens rotation and Householder transformation [45], but open questions remain 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 [1, 2] that use cosine-sine decomposition. The cosine-sine decomposition was first used by Tucci [53] for compilation purposes. In practice, commercial compilers ubiquitously deploy only KAK [5] 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; Shende et al. [2] mention a factor of 9, improved by Iten et al. [1] to 4×. The published approaches are hard to extend to different qubit gate sets, however, and it remains to be seen whether they can handle qutrits.\(^4\)

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 that of Martinez et al. [3], who use 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, whereas we show a more regimented way. When Martinez et al. [3] describe their results, they claim that Mølmer–Sørensen counts are directly comparable to CNOT counts. By this metric, we seem to generate circuits comparable to or shorter than theirs. It is not clear how their approach behaves when topology constraints are

\(^4\)Vitanov [54] describes a method using Givens rotations and Householder decomposition.
present. The direct comparison is further limited by 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 in the work of
Khatri et al. [4]. The main contribution is to use a quantum annealer to do searches over sequences
of increasing gate depth. The authors 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. [4] for two-qubit systems and Martinez
et al. [3] 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 criterion, as many algorithms can be viewed as
either. For example, the divide-and-conquer algorithm meet-in-the-middle proposed in the work
of Amy et al. [41], although designed for exact circuit synthesis, may also be used to construct an
$\epsilon$-approximate circuit. The results seem to indicate that the algorithm failed to synthesize a three-
qubit QFT circuit. 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.

11 CONCLUSION

In this article, we describe the LEAP compiler and modifications to a search and numerical opti-
mization based synthesis algorithm. The results indicate that we can empirically provide optimal-
depth circuits in a topology-aware manner for programs up to six qubits. The techniques employed
prefix formation, incremental resynthesis, dimensionality reduction, and multitstart optimization
and can be easily generalized to other algorithms from this class. We believe LEAP provides the
best-quality optimizer currently available for circuits up to six qubits on NISQ hardware. Further-
more, LEAP is the linchpin in our scalable synthesis algorithms (QGo [31], QuEst [32]) using circuit
partitioning techniques. With these algorithms, we have demonstrated the synthesis of circuits up
to hundreds of qubits. LEAP has been released as part of the BQSKit (Berkeley Quantum Synthesis
Toolkit) infrastructure.

REFERENCES

[1] Raban Iten, Roger Colbeck, Ivan Kukuljan, Jonathan Home, and Matthias Christandl. 2016. Quantum circuits for
isometries. Physical Review A 93 (March 2016), 032318. DOI: http://dx.doi.org/10.1103/PhysRevA.93.032318

[2] V. V. Shende, S. S. Bullock, and I. L. Markov. 2006. Synthesis of quantum-logic circuits. IEEE Transactions on Computer-
Aided Design of Integrated Circuits and Systems 25, 6 (June 2006), 1000–1010. DOI: http://dx.doi.org/10.1109/TCAD.2005.855930

[3] E. Martinez, T. Monz, D. Nigg, P. Schindler, and R. Blatt. 2016. Compiling quantum algorithms for architectures with
multi-qubit gates. New Journal of Physics 18, (July 2016), 063029. DOI: http://dx.doi.org/10.1088/1367-2630/18/6/063029

[4] Sumeet Khatri, Ryan LaRose, Alexander Poremba, Lukasz Cincio, Andrew T. Sornborger, and Patrick J. Coles. 2018.
Quantum-assisted quantum compiling. arXiv e-prints (July 2018), arXiv:1807.00800.

[5] Robert R. Tucci. 2005. An introduction to Cartan’s KAK decomposition for QC programmers. arXiv e-prints (July 2005),
arXiv:quant-ph/quant-ph/0507171.

[6] M. G. Davis, E. Smith, A. Tudor, K. Sen, I. Siddiqi, and C. Iancu. 2020. Towards optimal topology aware quantum circuit
synthesis. In Proceedings of the 2020 IEEE International Conference on Quantum Computing and Engineering (QCE’20).
223–234. DOI: http://dx.doi.org/10.1109/QCE49297.2020.00036

[7] Ed Younis, Koushik Sen, Katherine Yelick, and Costin Iancu. 2020. QFAST: Quantum synthesis using a hierarchical
continuous circuit space. arXiv:quant-ph/2003.04462.

[8] IBM Qiskit. n.d. Home Page. Retrieved August 12, 2022 from https://qiskit.org/.

[9] Seyon Sivarajah, Silas Dilkes, Alexander Cowtan, Will Simmons, Alec Edgington, and Ross Duncan. 2020. $|\text{ket}|$: A
retargetable compiler for NISQ devices. Quantum Science and Technology 6, 1 (Nov. 2020), 014003. DOI: http://dx.doi.
org/10.1088/2058-9565/ab8e92

[10] Raban Iten, Oliver Reardon-Smith, Luca Mondada, Ethan Redmond, Ravjot Singh Kohli, and Roger Colbeck. 2019.
Introduction to UniversalQCCompiler. arXiv e-prints (April 2019), arXiv:1904.01072.
[11] Victor Namias. 1980. The fractional order Fourier transform and its application to quantum mechanics. *IMA Journal of Applied Mathematics* 25, 3 (March 1980), 241–265. DOI: http://dx.doi.org/10.1093/imamat/25.3.241

[12] P. E. Hart, N. J. Nilsson, and B. Raphael. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics* 4, 2 (July 1968), 100–107. DOI: http://dx.doi.org/10.1109/TSSC.1968.300136

[13] Jeffrey Larson and Stefan M. Wild. 2018. Asynchronously parallel optimization solver for finding multiple minima. *Mathematical Programming Computation* 10, 3 (Feb. 2018), 303–332. DOI: http://dx.doi.org/10.1007/s12532-017-0131-4

[14] Sergey Bravyi, David Gosset, and Robert König. 2018. Quantum advantage with shallow circuits. *Science* 362, 6412 (2018), 308–311. DOI: http://dx.doi.org/10.1126/science.aar3106

[15] Jarrod R. McClean, Jonathan Romero, Ryan Babbush, and Alán Aspuru-Guzik. 2016. The theory of variational hybrid quantum-classical algorithms. *New Journal of Physics* 18, 2 (2016), 23023.

[16] Dongbin Shin, Hanno Hübener, Umberto De Giovannini, Hosub Jin, Angel Rubio, and Noeung Park. 2018. Phonon-driven spin-Floquet magneto-valleytronics in MoS₂. *Nature Communications* 9, 1 (2018), 638.

[17] Lindsay Bassman, Kuang Liu, Aravind Krishnamoorthy, Thomas Linker, Yifan Geng, Daniel Shebib, Shogo Fukushima, et al. 2020. Towards simulation of the dynamics of materials on quantum computers. *Physical Review B* 101, 18 (2020), 184305.

[18] Mario Motta, Chong Sun, Adrian T. K. Tan, Matthew J. O’Rourke, Erika Ye, Austin J. Minnich, Fernando G. S. L. Brandão, and Garnet Kin-Lic Chan. 2020. Determining eigenstates and thermal states on a quantum computer using quantum imaginary time evolution. *Nature Physics* 16, 2 (2020), 205–210. DOI: http://dx.doi.org/10.1038/s41567-019-0704-4

[19] D. Deutsch. 1989. Quantum computational networks. *Proceedings of the Royal Society of London A: Mathematical and Physical Sciences* 425, 1868 (Sept. 1989), 73–90. DOI: http://dx.doi.org/10.1098/rspa.1989.0099

[20] Vadym Kluschnikov, Alex Bocharov, and Krysta M. Svore. 2014. Asymptotically optimal topological quantum compiling. *Physical Review Letters* 112 (April 2014), 140504. DOI: http://dx.doi.org/10.1103/PhysRevLett.112.140504

[21] Nikolas Hansen. 2016. The CMA evolution strategy: A tutorial. *CoRR* abs/1604.00772.

[22] Dong C. Liu and Jorge Nocedal. 1989. On the limited memory BFGS method for large scale optimization. *Mathematical Programming* 45, 1–3 (1989), 503–528. DOI: http://dx.doi.org/10.1007/bf01589116

[23] Sameer Agarwal, Keir Mierle, and the Ceres Solver Team. 2022. Ceres Solver. Retrieved August 12, 2022 from http://ceres-solver.org.

[24] Laurens Van Der Maaten, Eric Postma, and Jaap Van den Herik. 2009. Dimensionality reduction: A comparative review. *Journal of Machine Learning Research* 10 (2009), 66–71.

[25] Vladislav Sovrasov. 2018. Comparison of dimensionality reduction schemes for derivative-free global optimization algorithms. *Procedia Computer Science* 136 (2018), 136–143. DOI: http://dx.doi.org/10.1016/j.procs.2018.08.246

[26] Stephen Hudson, Jeffrey Larson, Stefan M. Wild, David Bindel, and John-Luke Navarro. 2021. *libEnsemble Users Manual*. Technical Report Revision 0.7.2. Argonne National Laboratory.

[27] Stephen Hudson, Jeffrey Larson, John-Luke Navarro, and Stefan Wild. 2021. *libEnsemble*: A library to coordinate the concurrent evaluation of dynamical ensembles of calculations. *IEEE Transactions on Parallel and Distributed Systems* 33, 4 (2022), 977–988. DOI: http://dx.doi.org/10.1109/tpds.2021.3082815

[28] Ed Younis, Koushik Sen, Katherine Yelick, and Costin Iancu. 2021. QFAST: Conflating search and numerical optimization for scalable quantum circuit synthesis. arXiv:quant-ph/2103.07093.

[29] Heng Fan, Vwani Roychowdhury, and Thomas Szkopek. 2005. Optimal two-qubit quantum circuits using exchange circuit approximation. arXiv:quant-ph/2108.12714.

[30] Lindsay Bassman, Roel Van Beeumen, Ed Younis, Ethan Smith, Costin Iancu, and Wibe A. de Jong. 2021. Constant-depth circuits for dynamic simulations of materials on quantum computers. arXiv:quant-ph/2103.07429.

[31] Stuart J. Russell and Peter Norvig. 2021. *Artificial Intelligence: A Modern Approach* (4th ed.). Pearson, Hoboken, NJ.

[32] Vivek V. Shende, Igor L. Markov, and Stephen S. Bullock. 2004. Minimal universal two-qubit controlled-NOT-based circuits. *Physical Review A* 69, 6 (June 2004), 062321. DOI: http://dx.doi.org/10.1103/physreva.69.062321

[33] C. M. Dawson and M. A. Nielsen. 2005. The Solovay-Kitaev algorithm. *Quantum Information & Computation* 6, 1 (2005), 81–95.

[34] A. B. Nagy. 2016. On an implementation of the Solovay-Kitaev algorithm. arXiv:quant-ph/0606077.
[38] O. Al-Ta’Ani. 2015. *Quantum Circuit Synthesis Using Solovay-Kitaev Algorithm and Optimization Techniques*. Ph.D. Dissertation. Kansas State University.

[39] A. De Vos and S. De Baerdemacker. 2016. Block-ZXZ synthesis of an arbitrary quantum circuit. *Physical Review A* 94, 5 (Nov. 2016), 052317. DOI: http://dx.doi.org/10.1103/PhysRevA.94.052317

[40] Alex Bocharov and Krysta M. Svore. 2012. Resource-optimal single-qubit quantum circuits. *Physical Review Letters* 109, 19 (Nov. 2012), 190501. DOI: http://dx.doi.org/10.1103/PhysRevLett.109.190501

[41] Matthew Amy, Dmitri Maslov, Michele Mosca, and Martin Roetteler. 2013. A meet-in-the-middle algorithm for fast synthesis of depth-optimal quantum circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 32, 6 (June 2013), 818–830. DOI: http://dx.doi.org/10.1109/TCAD.2013.2244643

[42] Esteban A. Martinez, Thomas Monz, Daniel Nigg, Philipp Schindler, and Rainer Blatt. 2016. Compiling quantum algorithms for architectures with multi-qubit gates. *New Journal of Physics* 18, 6 (2016), 063029.

[43] B. Giles and P. Selinger. 2013. Exact synthesis of multiqubit Clifford+T circuits. *Physical Review Letters* 87, 3 (March 2013), 032332. DOI: http://dx.doi.org/10.1103/PhysRevA.87.032332

[44] S. S. Bullock and I. L. Markov. 2003. An arbitrary two-qubit computation in 23 elementary gates or less. In *Proceedings of the 2003 Design Automation Conference (DAC’03)*. 324–329. DOI: http://dx.doi.org/10.1109/DAC.2003.1219017

[45] J. Urias. 2010. Householder factorization of unitary matrices. *Journal of Mathematical Physics* 51 (2010), 072204.

[46] Mikko Möttönen, Juha J. Vartiainen, Ville Bergholm, and Martti M. Salomaa. 2004. Quantum circuits for general multiqubit gates. *Physical Review Letters* 93, 13 (Sept. 2004), 130502. DOI: http://dx.doi.org/10.1103/PhysRevLett.93.130502

[47] M. Amy and M. Mosca. 2016. T-count optimization and Reed-Muller codes. arXiv:1601.07363v1.

[48] G. Seroussi and A. Lempel. 1980. Factorization of symmetric matrices and trace-orthogonal bases in finite fields. *SIAM Journal on Computing* 9, 4 (1980), 758–767. DOI: http://dx.doi.org/10.1137/0209059 arXiv:https://doi.org/10.1137/0209059

[49] Neil J. Ross. 2015. Optimal Ancilla-free Clifford+V approximation of Z-rotations. *Quantum Information & Computation* 15, 11-12 (Sept. 2015), 932–950. http://dl.acm.org/citation.cfm?id=2871350.2871354.

[50] V. Kliuchnikov, D. Maslov, and M. Mosca. 2016. Practical approximation of single-qubit unitaries by single-qubit quantum Clifford and T circuits. *IEEE Transactions on Computers* 65, 1 (Jan. 2016), 161–172. DOI: http://dx.doi.org/10.1109/TC.2015.2409842

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

[52] Adam Paetznick and Krysta M. Svore. 2014. Repeat-until-success: Non-deterministic decomposition of single-qubit unitaries. *Quantum Information & Computation* 14, 15–16 (Nov. 2014), 1277–1301.

[53] Robert R. Tucci. 1999. A rudimentary quantum compiler. *arXiv e-prints* (Feb. 1999), arXiv:quant-ph/9902062.

[54] Nikolay Vitanov. 2012. Synthesis of arbitrary SU(3) transformations of atomic qutrits. *Physical Review A* 85 (March 2012), 032331. DOI: http://dx.doi.org/10.1103/PhysRevA.85.032331

Received 21 June 2021; revised 3 June 2022; accepted 15 June 2022.