LEAP: Scaling Numerical Optimization Based Synthesis Using an Incremental Approach

Ethan Smith, Marc G. Davis
University of California, Berkeley
ethanhs@berkeley.edu,marc.davis@berkeley.edu

Jeffrey M. Larson
Argonne National Laboratory
jmlarson@anl.gov

Ed Younis, Costin Iancu
Lawrence Berkeley National Laboratory
edyounis@lbl.gov, cciancu@lbl.gov

***Abstract*—We present techniques for incremental synthesis of quantum circuits using numerical optimization and search.**

Due to its ability to produce short depth circuits, synthesis provides a very valuable tool for quantum circuit optimization, especially for Noisy Intermediate Scale Quantum devices with short coherence time and noisy gates.

The techniques shown [1], [2] to produce shortest depth circuits use a combination of numerical optimization (of parameterized gate representations) and search over circuit structures. While providing close to optimal depth, they face scalability challenges:

- The number of parameters to optimize grows with circuit depth to the point where optimizers can't solve.
- The number of intermediate solutions to consider is exponential in the number of links/qubits.
- The objective function for optimization is complex and optimizers often get stuck in local minima.

In this paper we present the LEAP (Larger Exploration by Approximate Prefixes) compiler to scale synthesis along these three dimensions using an incremental approach. As the baseline we use Qsearch [1], an optimal depth synthesis algorithm which successfully handles circuits up to four qubits.

Qsearch optimal depth synthesis: The approach taken by Qsearch is to frame the question of optimal gate synthesis as a combinatorial optimization problem. Given a universal gate set for a quantum computer, Qsearch permutes combinations of gates (also called "layers") at each link placement possibility, building on the previous best placements to form a circuit structure. A numerical optimizer is run on the circuit structure to produce a score. The score guides the A* search algorithm towards the structure and parameters which minimizes the difference between the generated circuit and the input unitary based on a distance metric derived from the Hilbert-Schmidt norm.

Due to scalability limitations already mentioned, Qsearch can only feasibly synthesize up to four qubit input unitaries. In addition, Qsearch may miss the best solution due to the numerical optimizer missing the global optimum. These are common limitations of synthesis algorithms based on numerical optimization.

This work was supported by the Advanced Quantum 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.

I. Methods

Incremental Synthesis: Qsearch works by extending a circuit with one CNOT layer at a time and evaluating the potential partial solutions to lead to an optimal solution. For each partial solution a full numerical optimization step is performed.

LEAP was inspired by experience with the behavior of Qsearch and the structure of the problem. During the search, Qsearch would sometimes make significant progress in score very quickly, while other times it would slowly search a large number of nodes and make little progress. Generally, after a certain amount of progress, little to no backtracking occurred. By choosing a single structure as a prefix after significant progress is made, the search space can be greatly reduced. The challenge is to determine a partial solution in a manner that does not change the overall trajectory of the search.

LEAP operates very similarly to Qsearch. To start, a layer of single qubit gates (e.g. U3s) are placed on all qubit lines. This forms the root node and the original prefix. New layers are added to the search tree by placing 2-qubit parameterizations at each link. For the root node and each successive node in the search tree, a numerical optimizer is used to find the best parameterization to approximate the input unitary. This forms the search tree. The search of the tree is exactly the same as Qsearch, except previous best scores are tracked in a list. For each node that has the new best score, we compare it to a linear regression of previous scores. If it is a significant improvement over the projected score, we can use the current circuit as a prefix. However, we noticed this can lead to many extra layers being added superfluously, so the algorithm also checks that a minimum amount of searching has been done, to make sure LEAP isn't just adding and undoing work. We then re-start the search using our new prefix as the root node, keeping track at what depth of layers we re-fixed the prefix. Once the score falls below the desired threshold, the search stops and the result is returned.

Reoptimization: The end result of incremental synthesis (or other divide-and-conquer, partitioning, etc. techniques) is that circuit pieces are optimized in disjunction, with the potential of missing the optimal solution. The basic observation here is that by chopping and combining pieces of the final LEAP compiled circuit, we can create new, unseen circuits for the optimization process. Once a result is produced by LEAP, the output circuit and the list of depths where prefixes were fixed is

			Qsearch			QFAST †			LEAP with Re-optimization		
ALG	Qubits	Ref	CNOT	Unitary Distance	Time (s)	CNOT	Unitary Distance	Time (s)	CNOT	Unitary Distance	Time (s)
QFT	4	12	14	$6.7 * 10^{-16}$	2429.387881	24	$8.0 * 10^{-13}$	119.7	13	$6.7 * 10^{-16}$	77.9
TFIM-22	4	126	12	$1.2 * 10^{-15}$	2450.323824	46	$1.1 * 10^{-8}$	349.3	12	$7.8 * 10^{-16}$	41.7
TFIM-95	4	570	12	$6.7 * 10^{-16}$	1221.453431	41	$2.5 * 10^{-12}$	717.2	12	$2.2 * 10^{-16}$	38.2
VQE	4	91							21	$6.5 * 10^{-11}$	1662.4
full adder	4	N/A							18	0.0	115.3
HLF*	5	13							13	0.0	117.8
MUL	5	17							13	$1.1 * 10^{15}$	425.6
QFT	5	20				45	$5.3 * 10^{-13}$	373.9	28	$5.9 * 10^{-15}$	3154.1
TFIM-40	5	320				130	$1.8 * 10^{-12}$	1910.3	20	$7.8 * 10^{-16}$	646.6
TFIM-100	5	800				280 ‡	$2.6 * 10^{-12}$ ‡	N/A	20	$2.1 * 10^{-15}$	462.7
TFIM-24	6	240				180 ‡	$1.6 * 10^{-5}$ ‡	N/A	28	$7.3 * 10^{-11}$	106604.8
TFIM-51	6	510				278 ‡	$1.5 * 10^{-5}$ ‡	N/A	31	$8.9 * 10^{-11}$	46609.9

Fig. 1. *Summary of synthesis results for Qsearch, QFAST, and LEAP. On average, LEAP produces the same or shorter results than Qsearch and QFAST in less time. * HLF is the Hidden Linear Function algorithm from [3]. †For 4 qubits the KAK backend was used, for 5 qubits the Qsearch backend was used for QFAST. ‡Results from the QFAST paper [2] using the UQ backend.*

passed to the re-optimizer. The re-optimizer removes segments to create "holes" of a size provided by the user centered on the points where LEAP fixed the prefix of the rest of the circuit, as this is where extra CNOTs are most often added. The re-optimizer then uses normal Qsearch with a higher quality but slower multi-start solver based on [4], but instead of inserting layers at the end of the circuit, it inserts them at the start of the hole. The search stops once a node with a score below the threshold is found, or the search fails to find a best node below the size of the hole. Then the next hole is formed and re-optimized, and this process is repeated until there are no more points where LEAP created a prefix.

Multi-start optimization: When evaluating the performance of several optimizers for Qsearch, we wanted to find how often the optimizers found the true minimum, as this has a significant impact on the optimality of Qsearch based algorithms. We evaluated the commonly used Google's Ceres [5] and an L-BFGS [6] implementation against the mulit-start APOSMM [4] optimizer. In our test, we took a 4 qubit QFT solution and ran the optimizers 100 times to evaluate how often they found the true minimum. The L-BFGS optimizer found a solution just 2% of the runs. Google's Ceres solver found a solution about 10% of the time. With 24 starting points, APOSMM found a solution 99% of the time. Therefore when using LEAP, we use Ceres because it is fast and scales well, and a missed solution will be found during re-optimization. During re-optimization, APOSMM is used, as it is much more likely to find true minimums, thus strengthening the optimality of search based algorithms.

II. Evaluation

To evaluate LEAP, we implemented the algorithm via modification to Qsearch and benchmarked with `Python` 3.8.2, using `numpy` 1.19.1 and in `Rust` 1.44.0. Software is available at `https://github.com/WolfLink/qsearch`. The tests were run on a server with a 4.7 GHz Ryzen 9 3950X with 16 cores and 32 threads. We considered well known algorithms such as the Variational Quantum Eigensolver (VQE), important building blocks such as the Quantum Fourier Transform, and physical simulation circuits such as the Transverse Field Ising Models (TFIM). As can be seen in Fig 1, LEAP with re-optimization is able to reduce the CNOT count by up to 48x, or 11x on average.

III. Conclusion

In this paper, we describe the LEAP compiler, modifications to the Qsearch algorithm which scale the compiler to be capable of synthesizing 4, 5, and some 6 qubit input unitaries and a re-optimizer which further reduces the number of 2-qubit parameterizations needed in the synthesized circuit. Compared to similar state of the art synthesis tools, such as Qsearch and QFAST, LEAP with our re-optimization strategy is able to compile 4 qubit unitaries up to 59x faster than Qsearch. LEAP is also able to synthesize 5 and 6 qubit unitaries with up to 14x fewer CNOTs compared to QFAST. On average, LEAP with re-optimization is able to reduce the number of CNOTs compared to the reference circuit by 11x. We believe our techniques can be easily generalized or transferred to other algorithms based on search of circuit structures.

References

[1] M. G. Davis, E. Smith, A. Tudor, K. Sen, I. Siddiqi, and C. Iancu, "Heuristics for quantum compiling with a continuous gate set," 2019.

[2] E. Younis, K. Sen, K. Yelick, and C. Iancu, "Qfast: Quantum synthesis using a hierarchical continuous circuit space," 2020.

[3] S. Bravyi, D. Gosset, and R. König, "Quantum advantage with shallow circuits," *Science*, vol. 362, no. 6412, p. 308–311, Oct 2018. [Online]. Available: http://dx.doi.org/10.1126/science.aar3106

[4] J. Larson and S. M. Wild, "Asynchronously parallel optimization solver for finding multiple minima," *Mathematical Programming Computation*, vol. 10, no. 3, 2 2018.

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

[6] R. H. Byrd, P. Lu, J. Nocedal, and C. Zhu, "A limited memory algorithm for bound constrained optimization," *SIAM J. Sci. Comput.*, vol. 16, no. 5, p. 1190–1208, Sep. 1995. [Online]. Available: https://doi.org/10.1137/0916069