Synthesis of Approximate Parametric Circuits for Variational Quantum Algorithms

Blake Burgstahler[1], Ellis Wilson[1], Scott Pakin[2], Frank Mueller[1]
fmuelle@ncsu.edu
[1] North Carolina State University, [2] Los Alamos National Laboratory

Abstract

This work presents a novel approach to synthesize approximate circuits for the ansatze of variational quantum algorithms (VQA) and demonstrates its effectiveness in the context of solving integer linear programming (ILP) problems. Synthesis is generalized to produce *parametric* circuits in close approximation of the original circuit and to do so *offline*. This removes synthesis from the (online) critical path between repeated quantum circuit executions of VQA. We hypothesize that this approach will yield novel high fidelity results beyond those discovered by the baseline without synthesis. Simulation and real device experiments complement the baseline in finding correct results in many cases where the baseline fails to find any and do so with on average 32% fewer CNOTs in circuits.

1 Introduction

Variational quantum algorithms (VQAs) [11, 14] are a promising approach to solve relevant quantum chemistry [6, 14] and optimization problems algorithmically [4] on quantum computers. A VQA comprises a classical loop that repeatedly executes a parameterized quantum kernel alternating with classical optimization on the kernel's output to identify new parameters to consider.

VQAs, like other quantum algorithms, are limited in practice on contemporary hardware due to their sensitivity to noise, such as stray particles striking the system and disrupting the quantum state and leading to decoherence. The longer a program runs, the more susceptible it is to noise and to producing erroneous output. Because quantum computers typically observe higher error rates for two-qubit gates than for single-qubit gates, quantum algorithms are best expressed with circuits using as few two-qubit gates as possible. This constrains their application to the modeling of only small molecular systems or small optimization problems in terms of both ansatz and problem Hamiltonians.

Circuit-synthesis approaches [2, 19, 21, 22] can find shorter circuits with fewer two-qubit gates that approximately implement a given Hamiltonian, but circuit synthesis is a very time-consuming process. Synthesis, in essence, is a nonlinear optimization problem that aims to approximate a given fixed circuit, *not* a generalized parametric circuit. Because VQAs are iterative and use different parameters (single-qubit rotation angles) each iteration, invoking circuit synthesis on a per-iteration basis would dominate any performance gains from the use of a quantum computer.

The state of the art in integrating VQA with circuit synthesis requires parameter optimization and circuit generation to be applied in series [2, 13]: After the VQA parameters are optimized, an approximation is synthesized with an objective of reducing both the depth of the synthesized circuit and the number of two-qubit gates. This procedure repeats for each iteration, and the synthesized circuit is discarded after each iteration. Therefore, the major questions we ask are: 1. If synthesis across iterations were to reveal a common pattern for the ansatz, how could such knowledge be exploited? 2. Can this common pattern be found after a handful of iterations and result in fewer CNOTs (and therefore less susceptible to noise) circuits than naive gate replacement/substitution rules without incurring the cost of synthesis at *every* iteration?

This work develops a novel approach for VQAs with the ability to produce circuits with fewer CNOTs than naive gate replacement rules and by attempting to find and exploit a new QAOA ansatz pattern across iterations without incurring the cost of synthesis at every iteration. By exploiting synthesis, a latent parametric structure of the ansatz is sought. This parametric ansatz is instantiated at run time with low cost overhead between VQA iterations, effectively taking synthesis out of the run-time loop. This makes it feasible to dedicate a quantum device to a VQA execution across all iterations, as both classical optimization and ansatz instantiation impose low overheads as opposed to circuit synthesis.

Our work demonstrates the feasibility of such an approach by offline synthesis of approximate and parametric circuits for the ansatz of VQA. The proposed approach is implemented in the context of an existing quantum-programming framework, NchooseK [20]. Experiments assess our approach's effectiveness in simulation with and without noise and also on physical quantum devices. Our work makes the following contributions: 1. Circuit synthesis is generalized to produce *parametric* circuits in close approximation of the original circuit. 2. Synthesis is performed *offline*, and the resulting circuit replaces the QAOA ansatz, removing it from the critical path. 3. Simulation and real device experiments result in novel results beyond those discovered by the baseline without synthesis, i.e., our approach produces valid solutions when the baseline could not. 4. These benefits reduce CNOT counts by up to 45% with an average reduction of 32%.

2 Background

For brevity, we omit much of the basics of the gate model of quantum computing introduced by Deutsch [3] and include only some of the most relevant pieces to this work. Please reference the cited works for additional details on the ommitted basics.

2.1 Quantum Computing Basics

A single-qubit gate is encoded as a 2×2 matrix. In its most general form it can be represented as a continuous function of three parameters:

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

For example, the *Hadamard gate* is represented by

$$H = \mathrm{U3}(\tfrac{\pi}{2},0,\pi) = \tfrac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} . \tag{2}$$

Two-qubit gates are encoded as 4×4 matrices. For example, the controlled-NOT (CNOT) gate flips the target qubit iff the control qubit has value $|1\rangle$. CNOT has the following unitary representation:

$$\mathrm{CNOT} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \tag{3}$$

In short, a quantum program inputting and outputting n qubits can be represented by a single $2^n\times 2^n$ complex matrix. The power of a quantum computer is that it does not need to store nearly that much data.

A practical challenge is how to express a quantum program or subroutine in terms of single- and two-qubit gates, typically drawn from a hardware-specific set. Ideally, this decomposition should require as few time steps and as few two-qubit gates as possible. Although finding an optimal decomposition is NP-hard, Sec. 2.4 discusses heuristics for producing "good enough" quantum circuits.

2.2 Quantum Approximate Optimization Algorithm

The quantum approximate optimization algorithm (QAOA) is a VQA that attempts to solve combinatorial problems and was first proposed by Farhi, Goldstone, and Gutmann [4]. QAOA seeks optimal parameters $\{\beta,\gamma\}$ (each in turn a list of p values) for which some unitary $\mathbf{U}(\beta,\gamma)$ can be applied to an initial state to arrive at a final state that encodes the optimal solution.

This is achieved by decomposing $\mathbf{U}(\beta,\gamma)$ into two independent unitary matrices, $\mathbf{U}(\beta) = e^{-i\beta\mathbf{H_B}}$ and $\mathbf{U}(\gamma) = e^{i\gamma\mathbf{H_P}}$. The *problem Hamiltonian*, $\mathbf{H}_P$, implements a cost function whose minimum is sought. The *mixing Hamiltonian*, $\mathbf{H}_B$, represents a transition from one set of possible function-minimizing states to another such set.

The goal is to find β and γ parameters that minimize the expectation value $\langle\psi(\beta,\gamma)|\,\mathbf{H}_P\,|\psi(\beta,\gamma)\rangle$ (notation clarification: $\langle\phi| \equiv |\phi\rangle^\dagger$). The approach is to sample $\langle\psi(\beta,\gamma)|\,\mathbf{H}_P\,|\psi(\beta,\gamma)\rangle$ repeatedly on a quantum computer, then run a classical optimizer such as COBYLA [15] on the samples to propose new $\{\beta,\gamma\}$ values that are more likely to minimize the expectation value. The whole process repeats for a target number or iterations or until some convergence criterion is met. The QAOA loop is depicted as the blue flow in Fig. 2.

2.3 Quadratic Unconstrained Binary Optimization

One NP-hard problem that is well-suited to QAOA solution is the quadratic unconstrained binary optimization (QUBO) problem:

$$\underset{x\in\mathbb{B}^n}{\text{Minimize}}\; x^T Q x\,, \tag{4}$$

where $Q\in\mathbb{R}^{n\times n}$ and x is an n-dimensional binary vector, which maps conveniently to a quantum state. That is, x can be replaced with $|x\rangle$ in the objective function to yield $\langle x|\,Q\,|x\rangle$

2.4 Circuit Synthesis

Recall that quantum transformations (gates or entire circuits) acting on n qubits can be represented as a unitary matrix $\mathbf{U}\in\mathbb{C}^{N\times N}$ with $N=2^n$. Circuit synthesis aims to find a decomposition of $\mathbf{U}$ into small, unitary operators (typically 1- and 2-qubit gates) that, once appropriately combined, result in a circuit encoding the same transformation as $\mathbf{U}$. Many synthesis algorithms have been proposed [2, 13, 21, 22], each making different trade-offs among metrics such as execution time, circuit depth, 2-qubit gate count, and fidelity to the input circuit.

Synthesis methods use norm-based distance metrics to assess the similarity of $\mathbf{U}$ to its computed replacement $\mathbf{U}'$; the methods stop when the distance becomes sufficiently small. The distance metric can be as simple as $\|\mathbf{U}-\mathbf{U}'\|\le\epsilon$ for some tolerance ϵ. However, more recent approaches, such as QUEST [13] and most algorithms included in BQSKit [22], tend to use the Hilbert-Schmidt (HS) distance (Eq. (5)) as a stopping criterion. This distance is induced by the HS inner-product (Eq. (6)).

$$\Delta(\mathbf{U},\mathbf{U}') = \mathrm{HS}_{\mathrm{dist}}(\mathbf{U},\mathbf{U}') = \sqrt{1-\frac{\left\|\mathrm{Tr}\left(\mathbf{U}^\dagger\mathbf{U}'\right)\right\|^2}{N^2}} \tag{5}$$

$$\langle\mathbf{U},\mathbf{U}'\rangle_{\mathrm{HS}} = \mathrm{Tr}\Big(\mathbf{U}^\dagger\mathbf{U}'\Big) = \sum_i \lambda_i(\mathbf{U}^\dagger\mathbf{U}') \tag{6}$$

In general, the task of synthesis has exponential complexity in the number of qubits. This is remedied by partitioning the circuit into disjoint sub-circuits of capped width. These partitions (or blocks), once recombined, encode the same original circuit, as illustrated by Fig. 1. With partitioning, replacements for each block can be synthesized separately (and concurrently) and then stitched together to approximate the whole circuit. This process reduces computation time at the expense of reduced accuracy. If each block is $\Delta=\varepsilon_i$ then the total error of the full circuit is bounded above by the sum

of the errors, i.e., $\varepsilon_{\text{total}} \leq \sum_i \varepsilon_i$. In practice this bound has been found to be rather loose [21].

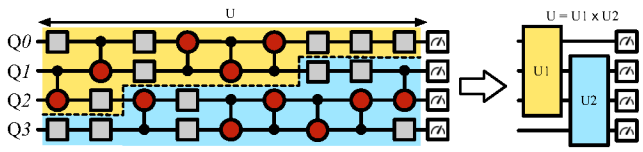


Figure 1. Partition a circuit U into U_1 and U_2, each acting on exactly 3 qubits. Image from: Patel et al. [13, Fig. 3].

2.5 Instantiation

Instantiation goes hand in hand with synthesis: rather than generating a new unitary/circuit the goal is to find values for a given circuit's parameterized gates (as in Eq. (1)) to approximate a given unitary. Like synthesis, instantiation attempts to minimize the HS distance between the two matrices:

$$\arg\min_{\alpha} \sqrt{1 - \frac{\left\|\text{Tr}\left(\mathbf{U}^{\dagger}\mathbf{C}(\alpha)\right)\right\|^2}{N^2}} \tag{7}$$

That is, instantiation serves to find the parameters α such that $\mathbf{C}(\alpha)\colon \mathbb{R}^k \mapsto \mathbb{C}^{N\times N}$ maximizes $\text{Tr}\left(\mathbf{U}^{\dagger}\mathbf{C}(\alpha)\right)$ for some unitary $\mathbf{U} \in \mathbb{C}^{N\times N}$.

2.6 NchooseK

NchooseK [20] is a constraint-based programming model whose implementation supports the quantum solution of a class of integer linear programming (ILP) problems without requiring any knowledge of quantum computing on the part of the user. Programs using this framework consist of Boolean variables and a set of constraints on them. Constraints take the form, "given a collection of Boolean variables N, exactly K of them must be *true*", and are notated $\text{nck}(N, K)$. K can be a set, in which case k variables must be *true* for some $k \in K$. Examples of how more complex constraints may be implemented are given by Wilson et al. [20].

NchooseK also supports *soft* constraints, expressed as $\text{nck}(N, K, \text{soft})$. The NchooseK solver will satisfy all hard (the default) constraints and as many soft constraints as possible. NchooseK has been used to encode a variety of challenging computational problems, including 3-SAT, minimum vertex cover, maximum cut, and map coloring [20].

Because NchooseK facilitates the expression of complex problems and because the implementation uses QAOA to solve these problems on gate-model quantum computers, it is a natural framework within which to implement our proposed techniques for accelerating VQAs on quantum computers.

3 Design

We now introduce our novel parameterized synthesis approach, which removes synthesis from the critical path (online) by performing all optimizations offline when creating a circuit ready for submission to quantum hardware (or simulators).

Our approach takes as input a parameterized circuit with parameters $\beta \in \mathbb{R}^p$, where p is the number of parameters. It then selects parameters to be assigned and applies a synthesis workflow to the circuit to produce a new parameterized circuit with parameters $\alpha \in \mathbb{R}^q$ assigned.[1] The process repeats with the new parameters. This leads to a set of synthesized circuits corresponding to the original circuit with particular parameters assigned. Once this set has grown to a satisfactory size, its contents can be sampled to select a template that is representative of the original circuit.

The template circuit is selected heuristically by first considering if any of the generated circuits have identical structure, favoring shallower circuits in the presence of multiple duplicates. If a structure is repeated, it is likely that it closely approximates the original parameterized circuit. If no structure is repeated, the circuit with the lowest two-qubit gate count is selected (favoring fewer single-qubit gates in the case of ties) in order to keep noise low. With the template selected, its parameters can be optimized and improved further. By selecting the template circuit in this manner, we attempt to create a circuit with fewer CNOTs than simply using gate replacement rules while still approximating the core QAOA ansatz.

Algorithm 1 extends QAOA with our parameterized-synthesis approach. It generates a set of synthesized circuits using the hybrid optimization scheme to guide the choice of initial parameters. For every improving choice of parameters we expect that the resulting synthesized circuits also will improve. Finally, the algorithm selects a template using the heuristic mentioned above and then optimizes the circuit, e.g., by applying standard QAOA techniques to optimize all parameters. This parametrized synthesis loop is depicted as the green flow in Fig. 2.

As mentioned previously, synthesis is computationally intensive. We attempt to reduce the number of circuits that need to be synthesized by adding a condition on the synthesis as shown in Algorithm 2. This condition checks if a previously synthesized circuits can be instantiated such that the HS distance between it and the original circuit is within some threshold. If so, the instantiated circuit is accepted as the template circuit for that iteration. In other words, new circuits are synthesized only when it is not possible to instantiate any existing circuit to sufficient accuracy. The instantiation loop is depicted as the red flow in Fig. 2.

[1] p and q are allowed to differ because each circuit may require a different number of parameters. Even if they require the same number of parameters, they may not have identical gate placements.

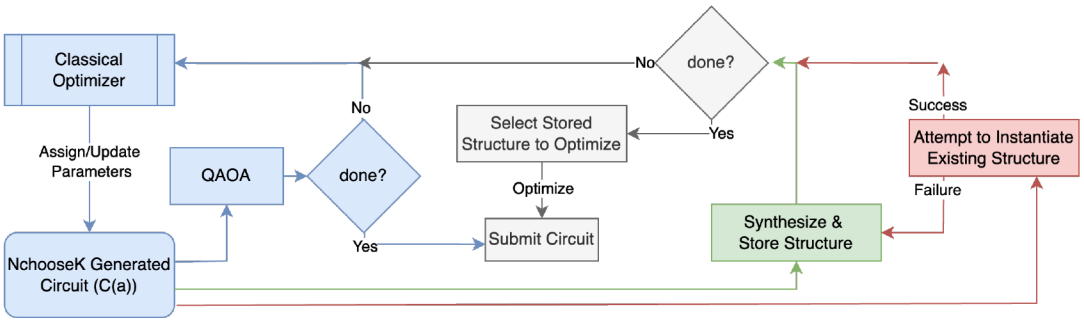


Figure 2. Flow chart representation of Base QAOA and Algorithms 1 and 2. The color scheme is standardized throughout the remainder of the Figures: Baseline/QAOA in blue, Synthesis in green, and Instantiation in red. It is worth noting that all methods are based on the core of QAOA and synthesis is inserted at each offline iteration of QAOA in order to generate a set of candidate structures to select from to replace the standard QAOA circuit.

Algorithm 1:
Parameterized Synthesis in a QAOA Context

Input: Parameterized circuit $|\psi(\beta,\gamma)\rangle$
$\beta,\gamma \leftarrow 1,1$
$\Psi \leftarrow \varnothing$
repeat
  Prepare $|\psi(\beta,\gamma)\rangle$ (assign parameters)
  Use synthesis to generate parameterized circuit $\psi(\alpha)$ $(\approx \psi(\beta,\gamma))$
  $\Psi \leftarrow \Psi \cup \{\psi(\alpha)\}$
  for *a suitably large number of repetitions* **do**
    Compute $\langle\psi(\alpha)|\, H_P\, |\psi(\alpha)\rangle$ on a quantum computer
  Find $\beta_{\text{new}}, \gamma_{\text{new}}$ with classical optimization
  $\beta,\gamma \leftarrow \beta_{\text{new}}, \gamma_{\text{new}}$
until β,γ *no longer improving*
Select $\psi_{\text{opt}}(\alpha)$ from Ψ
Apply QAOA to $\psi_{\text{opt}}(\alpha)$ to find optimal α
Output: Optimal circuit (fixed parameters)

Algorithm 2:
Parameterized Synthesis with Instantiation

Input: Parameterized circuit $|\psi(\beta,\gamma)\rangle$
$\beta,\gamma \leftarrow 1,1$
$|\psi_{\text{temp}}\rangle \leftarrow$ new synthesized circuit
$\Psi \leftarrow |\psi_{\text{temp}}\rangle$
repeat
  Prepare $|\psi(\beta,\gamma)\rangle$
  if $\exists \alpha_{opt}\ s.t.\ \Delta\left(|\psi(\beta,\gamma)\rangle, |\psi_{temp}(\alpha_{opt})\rangle\right) < 10^{-10}$
  then
    $\psi(\alpha) \leftarrow \psi_{\text{temp}}(\alpha_{\text{opt}})$
  else
    $\psi(\alpha) \leftarrow$ new synthesized circuit
  $\psi_{\text{temp}}(\alpha) \leftarrow \psi(\alpha)$
  ⋮
  $\beta,\gamma \leftarrow \beta_{\text{new}}, \gamma_{\text{new}}$
until β,γ *no longer improving*
Select $\psi_{\text{opt}}(\alpha)$ from Ψ
Apply QAOA to $\psi_{\text{opt}}(\alpha)$ to find optimal α
Output: Optimal Circuit (fixed parameters)

4 Implementation

We leverage existing software tools to implement Algorithms 1 and 2. First, NchooseK [12] serves as a means to generate problem-specific circuits and as a baseline against which to compare results for correctness. The Qiskit development kit [16] is used for execution of these circuits on simulators and on quantum hardware. Additionally, Qiskit provides a wrapper to the SciPy Python package for many classical optimization approaches, as well as many of its own implementations of optimizers. Finally, the Berkeley Quantum Synthesis Toolkit (BQSKit) [22] is leveraged to efficiently tackle both synthesis and instantiation problems while providing methods for converting to and from Qiskit-based circuits.

NchooseK generates a QUBO from user-specified constraints and then maps this QUBO to a parameterized quantum circuit for use as a QAOA problem Hamiltonian. We enhanced NchooseK as follows. First, we construct a parameterized quantum circuit with as many qubits as the QUBO has variables. A Hadamard gate is applied to every qubit, followed by phase rotations $R_Z(\beta)$ (single-qubit) and $R_{ZZ}(\beta)$ (two-qubit) gates according to the QUBO's linear and quadratic terms. Each gate is assigned a scale factor derived from the QUBO's coefficients. Finally, an $R_X(\gamma)$ gate is applied to each qubit.

A cost matrix $Q \in \mathbb{R}^{N\times N}$ is derived from the QUBO objective function such that that the function we aim to minimize

each iteration can be expressed as $\langle\psi| Q |\psi\rangle$ (the expected value of the circuit outputs). However, since ψ collapses upon measurement we approximate it by averaging over many ψ realizations (simulator or hardware shots). The generated circuit, together with the objective function to encode the problem Hamiltonian, serve as inputs to the baseline QAOA as well as our methods as described in Algorithms 1 and 2.

We create a customized synthesis workflow within BQSKit to control exactly how the synthesis is performed. This workflow is described by Algorithm 3. The algorithm first uses

Algorithm 3:
Implementation of Instantiation Algorithm

Given an NchooseK environment, generate initial parameterized circuit and associate objective
$\Psi \leftarrow \varnothing$
repeat
  Bind parameters to Qiskit circuit
  Convert circuit to BQSKit
  Attempt to instantiate existing template
  if *template exists AND instantiation within error bound* **then**
    Use the template
  else
    Use BQSKit to synthesize a new template circuit using CNOT and U3 gates
  Convert template to Qiskit
  $\Psi \leftarrow \Psi \cup$ template circuit
  Compute expected value of simulated counts
  Classically improve initial parameters
until *COBYLA optimizer terminates*
Heuristically select ψ_{opt} from Ψ
Classically optimize expected value of $\psi_{\text{opt}}(\alpha)$ to find optimal α
Output: Optimal circuit (fixed parameters)

BQSKit's QuickPartitioner to create the blocks necessary for improving synthesis time. This partitions the circuit by iterating over all gates and binning them together in a topological order [22]. Then, for each of these partitions the algorithm performs a pass of QSearch synthesis [2] followed by a gate-removal pass. In the gate removal pass, a gate is removed if the circuit can be instantiated within $\Delta = 10^{-10}$ of its initial state. Finally, the blocks are reconstructed into a full circuit with an "Unfold" pass. Additionally, prior to all of these passes, the algorithm employs a "Set Random Seed" pass to fix a random seed over all randomizers within any of the passes to ensure reproducibility. This also increases the likelihood that structurally identical subcircuits are generated that differ only by rotational angles. All other operations use existing functions provided by Qiskit and BQSkit.

5 Framework

In experiments to compare our parametric synthesis approach with non-parametric non-synthesis NchooseK/QAOA circuits, we utilize a subset of the benchmarks from Wilson et al. [20]. During experiments, the objective is to generate circuits that return valid solutions to these problems (both graph-based and satisfiability problems) with high probability using each of the three aforementioned approaches (baseline QAOA and Algorithms 1 and 2).

To this end, a number of input graphs are generated for solving respective graph problems. These graphs are generated using three vertices in fully connected subgraphs. As more of these subgraphs are added, at least two edges are added to connect the new subgraph to the exiting graph. In this way, we can generate graphs with vertices in any multiple of three and use them to create NchooseK environments for maximum cut (max cut) and minimum vertex cover (min vert cover) as specific test cases.

For maximum cut, we aim to produce a partition of the vertices such that we maximize the number of edges between the partitions. This is accomplished in NchooseK by encoding soft constraints such that adjacent vertices should be in unique sets. The solution is a partition that maximizes the satisfaction of the number of soft constraints.

For minimum vertex cover, we aim to produce the smallest set of vertices for which the set of all edges incident to these vertices includes all edges within the graph. This is encoded within NchooseK as a hard constraint that one or both vertices incident to each edge are selected. An additional soft constraint searches for the minimum subset of vertices that satisfies all hard constraints.

For both graph algorithms, each vertex in a graph corresponds to a qubit in the generated graph, and each edge constraint corresponds to a two-qubit interaction within the circuit.

Further, we consider a modified form of the 3-SAT satisfiability benchmarks also used in NchooseK. Namely, we create 3-SAT problems over a set of variables, which we vary in size from 3 to 7. From these variables, we generate 3-SAT constraints by randomly selecting 3 variables and assigning a negation to each of them with a probability of 0.5. Therefore, the qubit counts in these problems do not scale as directly as they did for the graph-based problems. For example, it is possible for the problem with 3 variables to have 9 qubits: one for each of the non-negated variables, one for each of the negated variables, and an ancilla for each constraint. By limiting the size of these problems, we ensure that the problems can fit on the target quantum computer.

Using this framework, we generate problems ranging from 3 to 21 qubits and compare the solution sets generated by our algorithms to the solutions generated by Microsoft's classical Z3 theorem prover [18]. However, we do not compare these results directly as Z3 provides only a *single* solution while

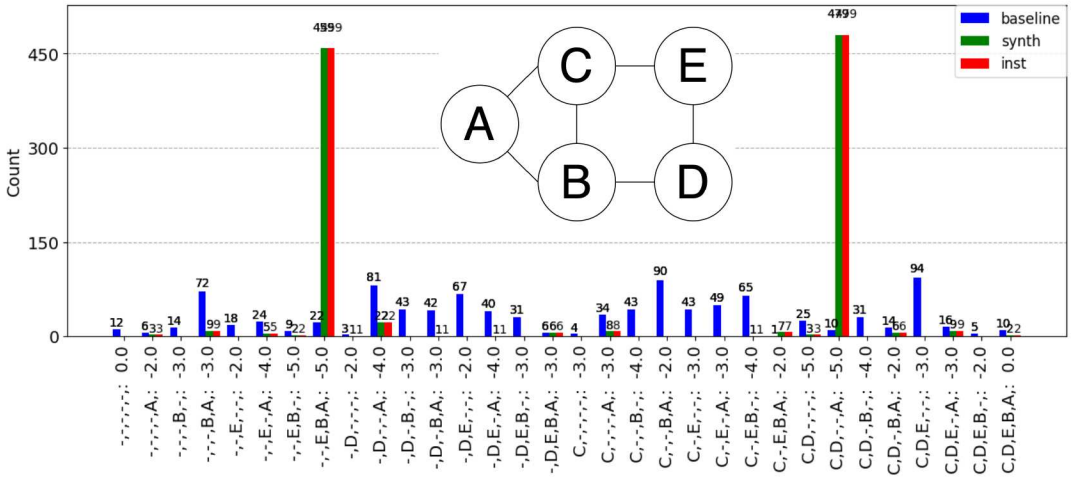


Figure 3. Histogram of counts returned for a 5 qubit max-cut problem whose corresponding graph is overlayed on the histogram. The results from 1024 shots on a simulator with a noise profile from the IBM Hanoi machine for each approach (baseline: typical QAOA [blue]; synthesis: as described in Algorithm 1 [green]; instantiation: as described in Algorithm 2 [red]) are shown. For this particular example, our synthesis based methods first produced 30 candidate structures. Each approach optimized its parameters for 1000 COYBLA iterations. Notice that our approaches heavily favor valid solutions while the baseline yields much more spread out (sub-optimal) results.

our generated circuits result in *multiple* solutions together with the approximated probability that each correct solution is found. With this in mind, we compare the *most likely* solutions generated to the solution provided by Z3 in two steps: 1. Is the candidate solution feasible? 2. Is the candidate solution "as good as" the Z3 solution?

The former is confirmed by validating that all hard constraints within the NchooseK environment are satisfied; the latter is confirmed by ensuring the solution satisfies at least as many soft constraints as the Z3 solution. This way, we never have to solve the underlying NP-hard QUBO created by NchooseK but rather verify generated solutions in linear time by comparing them in terms of their constraints. Furthermore, the above verification framework will be applied to results produced by the following executions of the generated circuits:

1. Ideal Simulation: using IBM's QASM Simulator;
2. Noisy Simulation: adding the hardware noise model from IBM's Hanoi machine to the simulator;
3. Real hardware: Transpile and submit to IBM's Hanoi machine.

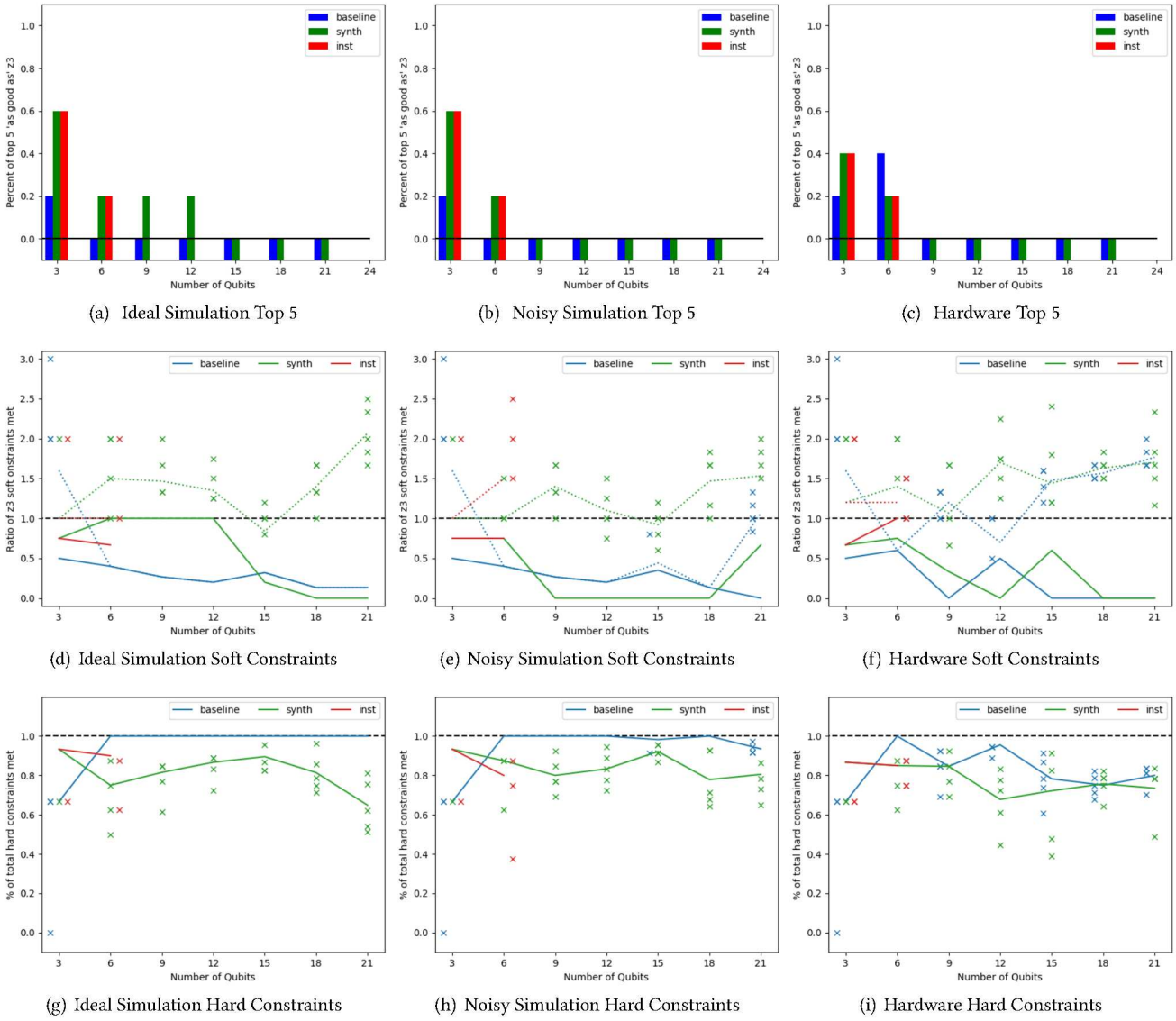
Ideal simulation is used to confirm theoretical differences between each of the approaches, while the introduction of noise in simulation is used to characterize the efficacy of each approach in practice. Finally, real hardware is used in experiments to show the efficacy of this offline circuit generation to improve results. For noisy simulation and hardware computations, we utilize the IBM Hanoi device, which hosts a 27 qubit Falcon r5.11 processor. Additionally, since the QASM simulator natively supports ZZ gates, in order to get a valid depth comparison to our synthesis approaches, we first transpile exclusively to CNOT and U3 gates.

6 Results

The first experiment features a max-cut problem of the graph depicted in the top-center of Fig. 3. Bars indicate the number of times (counts, y-axis) a certain cut in the graph was indicated in noisy simulation for (a) the baseline QAOA circuit generated by NchooseK (blue), (b) its approximate counterpart resulting from synthesis (green) and (c) the instantiated, parametric circuit (red).

The x-axis labels indicate the associated objective value (which is being minimized) as well as the vertices belonging to one partition. All other vertices lie in the other partition. We observe that the baseline approach finds not only the correct solutions but also many incorrect ones, many observed with a higher probability than the correct ones. Synthesis and instantiation peak at both correct solutions while tallying small counts for all incorrect solutions. This example illustrates the potential of our parametric approaches, and it demonstrates that the two methods both provide good results.

The next experiments focus on the benchmarks referenced in Sec. 5. Each set of benchmarks was allowed to run for up to 12 hours with each approach to complete as many cases as possible. In the cases of both the baseline QAOA and synthesis, all test cases completed and produced results. However, instantiation becomes intractably slow beyond 6 qubits due to scaling of the number of parameters involved in the optimization. For example, for a circuit instance with 100



(a) Ideal Simulation Top 5 (b) Noisy Simulation Top 5 (c) Hardware Top 5

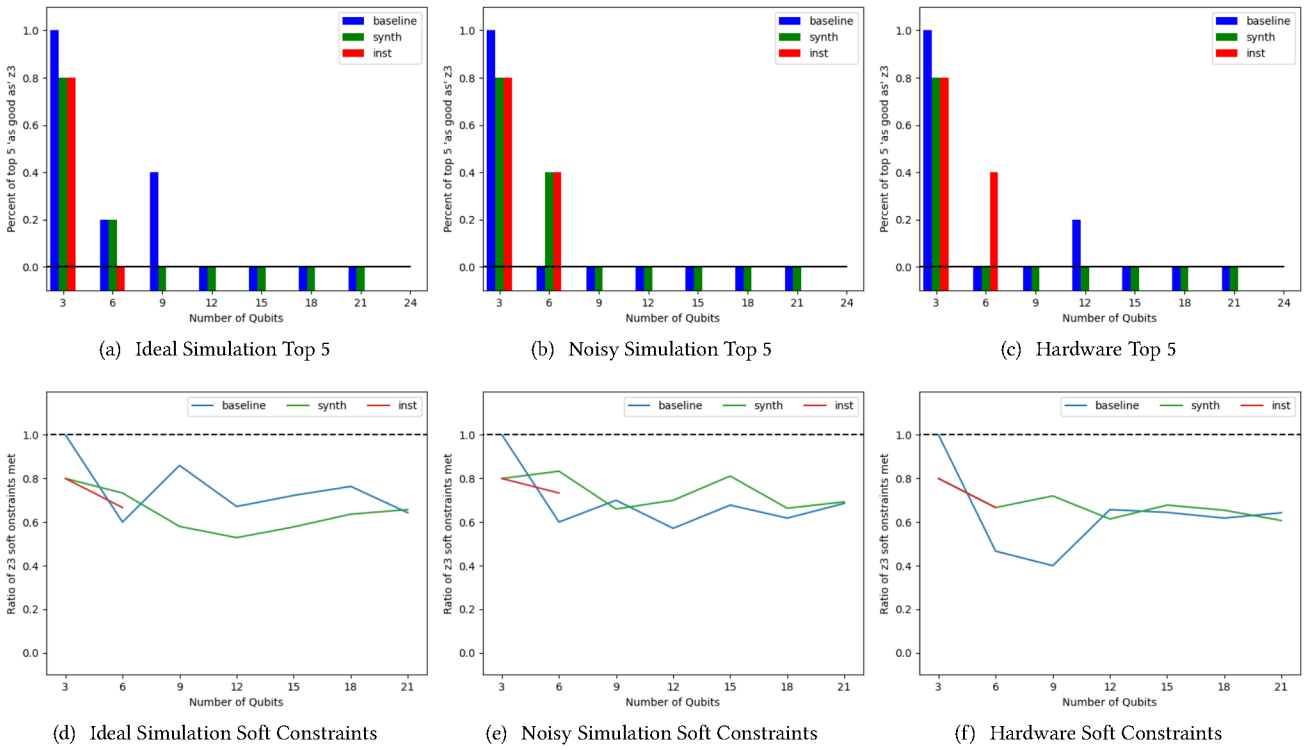
(d) Ideal Simulation Soft Constraints (e) Noisy Simulation Soft Constraints (f) Hardware Soft Constraints

(g) Ideal Simulation Hard Constraints (h) Noisy Simulation Hard Constraints (i) Hardware Hard Constraints

Figure 4. Minimum Vertex Cover with $p = 3$. x's mark solutions that failed to meet at least one hard constraint. The dotted lines include these x's in averages and the solid lines exclude them from the average. Instantiation results are omitted beyond 6 qubits as they did not finish within the time limit.

U3 gates, there will be 300 parameters that must be selected by the instantiation method, dramatically increasing the complexity of Eq. (7).

Fig. 4 depicts results for minimum vertex cover. Figs. 4(a) to 4(c) indicate via bar charts (top row) the probability (y-axis) of one of the top 5 solutions being "as good as" the result reported by the classical Z3 solver for different graph sizes, indicated by the number of qubits (x-axis). Any such result is first checked to confirm that (1) all hard constraints and (2) at least as many soft constraints as the classically found Z3 solution are met. Results are reported for (a) noise-free simulation (left), (b) noisy simulation (center), and (c) hardware execution (right). The horizontal line at $y = 0$ serves as a reference point as some experiments completed without any valid results, in particular for the baseline (blue). For the simulations in 4(a) and 4(b), we observe that the baseline QAOA circuits rapidly deteriorate at higher qubit counts, whereas our parametric methods consistently outperforms the baseline by improving fidelity significantly. While the baseline fails to find acceptable solutions, our methods consistently find them. When run on hardware in 4(c), we observe a different trend. We see improvements only for 3 qubits while the baseline outperforms others at 6 qubits. This



(a) Ideal Simulation Top 5 (b) Noisy Simulation Top 5 (c) Hardware Top 5

(d) Ideal Simulation Soft Constraints (e) Noisy Simulation Soft Constraints (f) Hardware Soft Constraints

Figure 5. Maximum Cut with $p = 3$ repetitions of its circuit. The line is the average over all cases. These problems only have soft constraints. Instantiation results beyond 6 qubits omitted as they did not complete within the time limit.

rapid falloff likely is caused by the increased complexity of vertex cover problems as they contain both hard and soft constraints.

Figs. 4(d) to 4(f) indicate if soft constraints are met (y-axis) relative to classical Z3 solving. Above 1.0, more constraints are met, and below 1.0, fewer are met. Lines average over the top 5 solutions, where dashed lines include and solid lines exclude any solutions that violate hard constraints (plotted as ×s). Often, both lines overlap (only solid visible). We observe that the synthesized and instantiated solutions are able to satisfy as many if not more soft constraints than the baseline. This is consistently the case for ideal and noisy simulation as well as for hardware runs.

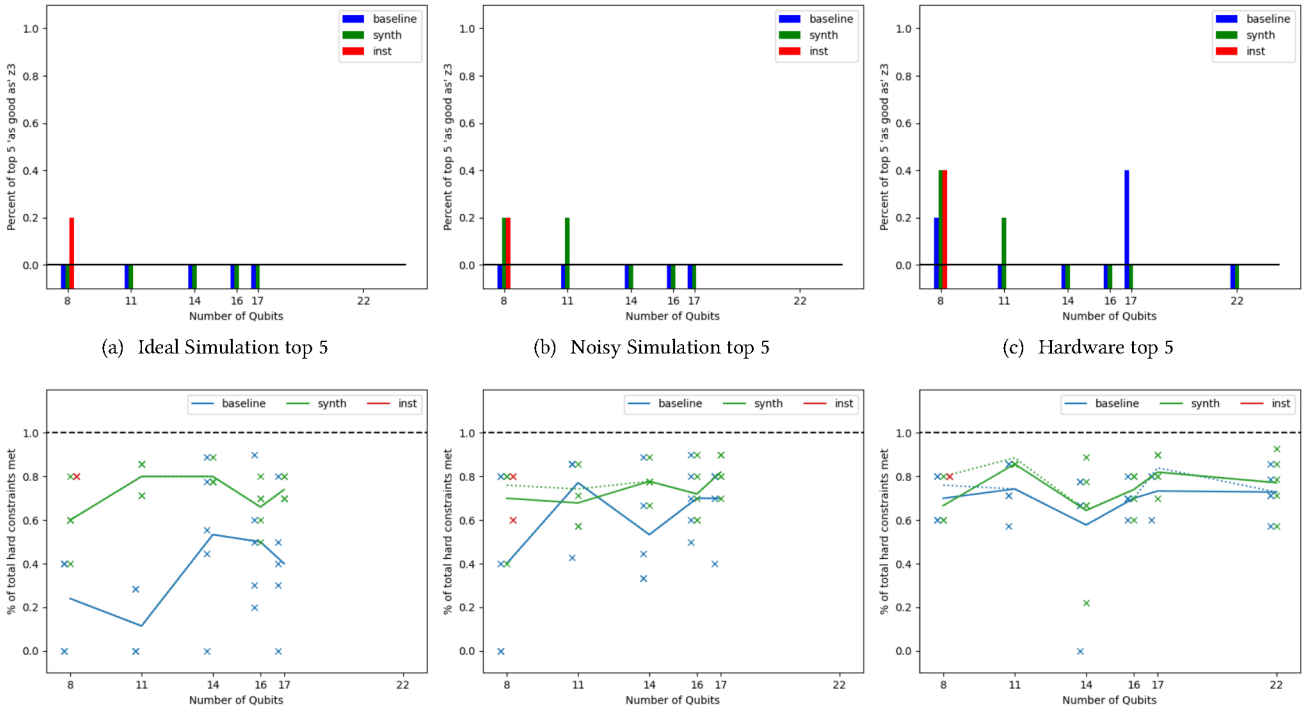
Figs. 4(g) to 4(i) plot the ratio of hard constraints (x-axis) in the same manner. In Fig. 4(g) we see that the baseline QAOA succeeds at satisfying hard constraints for all problems of 6 or more qubits while our methods are able to satisfy only 80% of hard constraints on average. We see similar trends in Fig. 4(h): the baseline consistently satisfies hard constraints while our approaches lag slightly behind at around 80% satisfaction. Finally, Fig. 4(i) shows hardware trends. Now the baseline begins not to consistently satisfy all hard constraints but still tends to satisfy more hard constraints than our methods.

Considering all the results, more hard constraints are broken by our parametric methods. Yet, by violating hard constraints we enable satisfaction of significantly higher numbers of soft constraints under simulation. On hardware, the baseline begins to break hard constraints as well. Given the current state of noisy quantum hardware, it is to be expected that hardware results are of lower fidelity than simulation.

Fig. 5 depicts results for max cut with similar graphs. Figs. 5(a) to 5(c) indicate results for the baseline up to 9 qubits (ideal) and a single results for 12 qubits (hardware). Our parametric methods consistently deliver correct results up to 6 qubits, even on hardware, but then do not fall off as fast as in the vertex cover, because max cut has reduced complexity and involves only soft constraints.

Max cut has only soft constraints. Figs. 5(d) to 5(f) indicate about the same number of soft constraints being broken across all methods compared to Z3 classical solving.

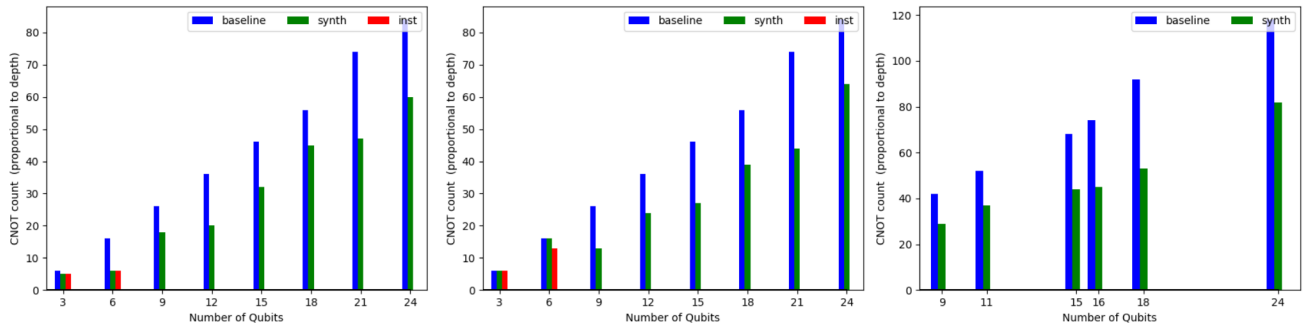
Fig. 6 depicts results for 3-SAT problems, which have only hard constraints. Figs. 6(a) to 6(c) indicate that our approaches outperform the baseline, except for an outlier on hardware at 17 qubits where the baseline delivers. Both Figs. 6(a) and 6(b) show that for lower qubit counts, our methods succeed in finding a valid solution while the baseline fails. Figs. 6(d) to 6(f) show that fewer hard constraints are being met by these methods than by Z3 yet more solutions



(a) Ideal Simulation top 5 (b) Noisy Simulation top 5 (c) Hardware top 5

(d) Ideal Simulation hard constraints (e) Noisy Simulation hard constraints (f) Hardware hard constraints

Figure 6. 3-SAT results for $p = 3$. These only include hard constraints and follow the same formatting as Fig. 4



(a) Minimum Vertex Cover (b) Maximum Cut (c) 3-Sat

Figure 7. CNOT count, a proxy for circuit depth, for each template circuit from each problem type.

are still being found by our approaches compared to the baseline.

Let us consider the percentage of hard constraints satisfied by the three methods. Since 3-SAT problems have only hard constraints, this directly gives us a measure of how close to the optimal solution we are. In each of Figs. 6(d) to 6(f) we see that the average of our methods trends closer to optimal across all test cases. In each plot, the solid line shows the average percentage of hard constraints satisfied for solutions that failed to satisfy all constraints. The dotted lines show the same average but also include solutions that satisfy the hard constraints as well. In an ideal simulation, our approaches satisfy more hard constraints on average in all cases. In noisy simulation and hardware runs, the same general trend continues, but the baseline outperforms our approaches at 11 qubits in noisy simulation and 17 qubits in the hardware runs.

Fig. 7 depicts the number of CNOT gates (y-axis) per number of qubits (x-axis), which is relevant for decoherence as CNOT count directly correlates to depth. We observe that our approaches produce circuits with the same or a reduced

number of CNOTs. At low qubit counts, our methods perform on par with the baseline. Beyond that, the benefits of our approach become clearer for circuits of larger qubit count, where we can see up to a 45% reduction in CNOT gate count and an average reduction of 32%.

Discussion: We believe that the absence of correct solutions at larger numbers of qubits stems from two main sources. First, at this scale we introduced enough partitions through synthesis that the compounding errors from each block may become significant as the number of qubits is increased. This effect compounds further when using repetitions of the synthesized ansatz in place of the standard QAOA ansatz and, unfortunately, seems to counteract any gains that may have been achieved by reducing CNOT counts. Prior works [2, 13] benchmark their synthesis approaches with some QAOA circuits, but neither use a QAOA circuit larger than 10 qubits so they have not established any precedent in terms of the scalability limitations of the baseline QAOA. Furthermore, since each of our approaches seeks to improve upon QAOA results by synthesizing approximations, it is reasonable to expect that if neither baseline QAOA yields valid solutions, nor will approximations of it especially when using a local search optimizer. In other words, when QAOA is close to a correct solution, our approach can improve upon it even further.

7 Related Work

7.1 Synthesis Methods

Our work relies on BQSKit [22] for circuit synthesis. Prior approaches using BQSKit exhibit a number of objectives. QUEST tries to reduce the CNOT gate count during synthesis to reduce noise [13]. QSearch [2] iterates over generated synthetic circuits using heuristics and the HS metric combined with subcircuit partitioning via QFAST. An algorithm for parameterized circuit instantiation is used to reduce noise in an optimization and gate-set retargeting approach [21]. Heuristics for reducing the number of CNOTs and considering topologies to avoid swaps are utilized via A* search for synthesis [1]. These instantiation and retargeting methods differ from our work in that we aim to find parametric circuits in an iterative manner, which removes synthesis from the critical path within VQA methods.

7.2 Circuit Parameterization

Previous works [7, e.g.] promote parameterized compilation techniques without the use of synthesis. Their techniques focus on reducing compilation time beyond the existing optimization levels available in Qiskit. Our work expands on this by introducing synthesis as the parameterization technique rather than manipulating the compiled (hardware ready) circuit.

Precompilation techniques [17] use a template that they adapt to a particular problem type and subsequently to a particular problem instance. Our templates are not formed as generic catch-alls for problem types, but rather are *specifically* generated for each instance via circuit synthesis.

7.3 NchooseK

NchooseK [12, 20] was designed to provide a domain-specific language to easily specify problems that can then be solved on classical computers, quantum annealers, and circuit-based quantum computers. It is based on the idea that a wide variety of constraints can be formulated as "K of these N Boolean values must be *true*". NchooseK enables the user to access the power of quantum computers solely through the expression of classical constraints. It relies on a QAOA-based quantum solver, but in the form used in the experiments it returns only the single most likely solution. Our work expands on NchooseK to focus on the generation of a circuit that results in a high probability for potentially *multiple* optimal solutions and very low probability for suboptimal solutions.

7.4 Other QAOA-based Work

The prevailing issue in QAOA is appropriate selection of the parameters. Some work [5, 9] considers ways to improve the formulation of the QAOA ansatz by iteratively fixing parameters or by introducing additional angles to optimize the ansatz. Their aim is to exploit structure as a means of improving performance. Other work explores gradient-free optimizers, which can handle the additional difficulties that come with the presence of noise and the requirement of a "black-box" evaluation of objective functions [8].

Work is being done to improve the performance of QAOA algorithms on hardware [10, 19]. The focus here is on optimal ways to map these problems onto hardware, namely by reducing the number of swaps necessary to execute the circuit while increasing parallelism.

Our work is orthogonal to these approaches as it concerns efficient QAOA circuit synthesis to improve noise resiliency while removing synthesis from the critical path between successive job executions of VQA methods.

8 Conclusions

We introduced and surveyed the current state of VQA approaches before presenting two novel circuit synthesis-based algorithms for producing quantum circuits that increase the probability of finding problem solutions. We show that these algorithms can be applied in the context of QAOA and confirm their viability over traditional QAOA. Generally, if QAOA finds a solution, our algorithms increase the likelihood of identifying either a better solution or an alternative one. Sometimes, our methods find solutions when baseline QAOA does not. Other times, our methods complement baseline QAOA with additional solutions. These results are all obtained with reduction in CNOT gate counts compared to the baseline when transpiled to the same gate set.

Acknowledgments

We would like to acknowledge contributions by Rohit Mohan to the initial proof of concept on small max-cut problems as a part of Dr. Mueller's Quantum Computing course. Research presented in this paper was supported by the Laboratory Directed Research and Development program at Los Alamos National Laboratory under project number 20210397ER. Los Alamos National Laboratory is operated by Triad National Security, LLC for the National Nuclear Security Administration of the U.S. Department of Energy (contract no. 89233218CNA000001). This work was also supported in part by LANL subcontract 725530 and by NSF awards PHY-1818914, PHY-2325080, MPS-2120757, CISE-2217020, and CISE-2316201. Released under LA-UR-24-28597.

References

[1] Marc G. Davis, Ethan Smith, Ana Tudor, Koushik Sen, Irfan Siddiqi, and Costin Iancu. Heuristics for quantum compiling with a continuous gate set. In *3rd International Workshop on Quantum Compilation as part of the International Conference On Computer Aided Design 2019*, Dec 2019.

[2] Marc G. Davis, Ethan Smith, Ana Tudor, Koushik Sen, Irfan Siddiqi, and Costin Iancu. Towards optimal topology aware quantum circuit synthesis. In *2020 IEEE International Conference on Quantum Computing and Engineering (QCE)*, pages 223–234, 2020.

[3] David Elieser Deutsch. Quantum computational networks. *The Royal Society London*, 425:73–90, September 8, 1989.

[4] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. A quantum approximate optimization algorithm, 2014.

[5] Rebekah Herrman, Phillip C Lotshas, James Ostrowski, Travis S. Humble, and George Siopsis. Multi-angle quantum approximate optimization algorithm. *Scientific Reports*, 12(6781), 2022.

[6] Abhinav Kandala, Antonio Mezzacapo, Kristan Temme, Maika Takita, Markus Brink, Jerry M Chow, and Jay M Gambetta. Hardware-efficient variational quantum eigensolver for small molecules and quantum magnets. *Nature*, 549(7671):242–246, 2017.

[7] A. M. Krol, K. Mesman, A. Sarkar, M. Möller, and Z. Al-Ars. Efficient parameterised compilation for hybrid quantum programming, 2022.

[8] Wim Lavrijsen, Ana Tudor, Juliane Müller, Costin Iancu, and Wibe de Jong. Classical optimizers for noisy intermediate-scale quantum devices. In *2020 IEEE International Conference on Quantum Computing and Engineering (QCE)*, pages 267–277, Oct 2020.

[9] Xinwei Lee, Yoshiyuki Saito, Dongsheng Cai, and Nobuyoshi Asai. Parameters fixing strategy for quantum approximate optimization algorithm. In *2021 IEEE International Conference on Quantum Computing and Engineering (QCE)*, pages 10–16, Oct 2021.

[10] Phillip C Lotshaw, Thien Nguyen, Anthony Santana, Alexander McCaskey, Rebekah Herrman, James Ostrowski, George Siopsis, and Travis S Humble. Scaling quantum approximate optimization on near-term hardware. *Scienticic Reports*, 2(12388), 2022.

[11] Nikolaj Moll, Panagiotis Barkoutsos, Lev S Bishop, Jerry M Chow, Andrew Cross, Daniel J Egger, Stefan Filipp, Andreas Fuhrer, Jay M Gambetta, Marc Ganzhorn, et al. Quantum optimization using variational algorithms on near-term quantum devices. *Quantum Science and Technology*, 3(3):030503, 2018.

[12] Scott Pakin and Ellis Wilson. NchooseK. https://github.com/lanl/NchooseK, 2023.

[13] Tirthak Patel, Ed Younis, Costin Iancu, Wibe de Jong, and Devesh Tiwari. QUEST: Systematically approximating quantum circuits for higher output fidelity. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '22, page 514–528, New York, NY, USA, 2022. Association for Computing Machinery.

[14] Alberto Peruzzo, Jarrod McClean, Peter Shadbolt, Man-Hong Yung, Xiao-Qi Zhou, Peter J Love, Alán Aspuru-Guzik, and Jeremy L O'brien. A variational eigenvalue solver on a photonic quantum processor. *Nature communications*, 5(1):4213, 2014.

[15] M. J. D. Powell. *A Direct Search Optimization Method That Models the Objective and Constraint Functions by Linear Interpolation*, pages 51–67. Springer, Dordrecht, Netherlands, 1994.

[16] Qiskit contributors. Qiskit: An open-source framework for quantum computing. https://qiskit.org/, 2023.

[17] Nils Quetschlich, Lukas Burgholzer, and Robert Wille. Reducing the compilation time of quantum circuits using pre-compilation on the gate level, 2023.

[18] Microsoft Research. Z3: an efficient theorem prover, 2023.

[19] Bochen Tan and Jason Cong. Optimal layout synthesis for quantum computing. In *Proceedings of the 39th International Conference on Computer-Aided Design*, ICCAD '20, New York, NY, USA, 2020. Association for Computing Machinery.

[20] Ellis Wilson, Frank Mueller, and Scott Pakin. Combining hard and soft constraints in quantum constraint-satisfaction systems. In *SC22: International Conference for High Performance Computing, Networking, Storage, and Analysis*, pages 161–174, November 2022.

[21] Ed Younis and Costin Iancu. Quantum circuit optimization and transpilation via parameterized circuit instantiation. In *2022 IEEE International Conference on Quantum Computing and Engineering (QCE)*, pages 465–475, Sep. 2022.

[22] Ed Younis, Costin C Iancu, Wim Lavrijsen, Marc Davis, and Ethan Smith. Berkeley quantum synthesis toolkit (BQSKit) v1. https://bqskit.lbl.gov/, April 2021.