Iterative layerwise training for the quantum approximate optimization algorithm

Xinwei Lee,[1,*] Xinjian Yan,[2] Ningyi Xie,[2] Dongsheng Cai,[3] Yoshiyuki Saito,[4] and Nobuyoshi Asai[5]

[1]*Graduate School of Systems and Information Engineering, University of Tsukuba, Ibaraki Prefecture 305-8577, Japan*
[2]*Graduate School of Science and Technology, University of Tsukuba, Ibaraki Prefecture 305-8577, Japan*
[3]*Faculty of Engineering, Information and Systems, University of Tsukuba, Ibaraki Prefecture 305-8577, Japan*
[4]*Graduate School of Computer Science and Engineering, University of Aizu, Fukushima Prefecture 965-0006, Japan*
[5]*School of Computer Science and Engineering, University of Aizu, Fukushima Prefecture 965-0006, Japan*

(Received 16 February 2024; accepted 15 April 2024; published 2 May 2024)

The capability of the quantum approximate optimization algorithm (QAOA) in solving combinatorial optimization problems has been intensively studied in recent years due to its application in the quantum-classical hybrid regime. Despite having difficulties that are innate in the variational quantum algorithms (VQA), such as barren plateaus and the local minima problem, QAOA remains one of the applications that is suitable for the recent noisy intermediate scale quantum (NISQ) devices. Recent works have shown that the performance of QAOA largely depends on the initial parameters, which motivate parameter initialization strategies to obtain good initial points for the optimization of QAOA. On the other hand, optimization strategies focus on the optimization part of QAOA instead of the parameter initialization. Instead of having absolute advantages, these strategies usually impose trade-offs to the performance of the optimization problems. One such example is the layerwise optimization strategy, in which the QAOA parameters are optimized layer by layer instead of the full optimization. The layerwise strategy costs less in total compared to the full optimization, in exchange of lower approximation ratio. In this work, we propose the iterative layerwise optimization strategy and explore the possibility for the reduction of optimization cost in solving problems with QAOA. Using numerical simulations, we found that by combining the iterative layerwise strategy with proper initialization strategies, the optimization cost can be significantly reduced in exchange for a minor reduction in the approximation ratio. We also show that in some cases, the approximation ratio given by the iterative layerwise strategy is even higher than that given by the full optimization.

DOI: 10.1103/PhysRevA.109.052406

I. INTRODUCTION

The quantum approximate optimization algorithm (QAOA) is a quantum-classical hybrid algorithm introduced in 2014 [1], aimed at solving combinatorial optimization problems on a universal quantum computer using the paradigm of discretized adiabatic quantum computation [2]. Many recent works have explored the potential of QAOA to have a quantum advantage in some of the optimization problems against their classical counterparts [3–7], and thus many variants of QAOA have been suggested to satisfy their respective needs [8–11]. However, QAOA inherits properties that are innate in the variational quantum algorithms (VQA), such as the existence of barren plateaus in the optimization landscape [12–15], and the exponential increase of local minima as the circuit depth increases [16,17], rendering the optimization of QAOA difficult. To tackle these problems, various initialization strategies are proposed so that QAOA can be initialized near the desired optima [5,18–20], aiding the convergence of the classical optimizers.

Besides the initialization strategies, optimization strategies are also essential in improving the optimization of QAOA. The layerwise strategy (or layerwise training) [21] is an optimization strategy which trains the QAOA parameters layer by layer. In layerwise training, only the latest parameters are updated while the other parameters are fixed as constant. This has led to a reduction in the optimization cost, but also causes the optimization to occur in a restricted search space, and the premature saturation (saturation before the approximation ratio reaches 1) of the layerwise training is reported in Ref. [21] for a projector Hamiltonian $H_z = |0\rangle\langle 0|$. We found that premature saturation also occurs in the max-cut Hamiltonian. However, layerwise training is shown to perform better in the quantum neural network [22].

In this work, we propose the iterative layerwise (abbreviated as ITLW) strategy, which improves the layerwise strategy and helps to prevent premature saturation. We found that the trainability of the parameters in ITLW is highly sensitive to the initial parameters used. Therefore, by combining ITLW with some initialization strategies, e.g., bilinear initialization [23] and Trotterized quantum annealing (TQA) initialization [18], we successfully reduce the optimization cost (evaluated by the total number of function calls) significantly in exchange for a minor reduction in the approximation ratio. The bilinear initialization uses information of parameters from previous depths to produce a high approximation ratio, while TQA directly initializes the parameters as a linear function at a specific depth (refer to Sec. II and Appendix A for details). For some small number of iterations $k = 2$, ITLW managed to

[*]xwlee@cavelab.cs.tsukuba.ac.jp

reduce the cost by almost half with a 4×10^{-3} reduction of the approximation ratio when combined with these initialization strategies.

The structure of the paper is summarized as follows. In Sec. II, we introduce the background of QAOA and the layerwise training. In Sec. III, we discuss the idea of the ITLW strategy and how the expected optimization cost scales with the QAOA circuit depth. In Sec. IV, we present and discuss the simulation results for ITLW combined with the initialization strategies (bilinear and TQA). Lastly, we conclude in Sec. V.

II. QAOA AND LAYERWISE TRAINING

The objective of QAOA is to maximize the expectation of a given cost Hamiltonian H_z with respect to the ansatz state $|\psi(\boldsymbol{\gamma}, \boldsymbol{\beta})\rangle$ prepared by the evolution of the alternating operators:

$$|\psi_p(\boldsymbol{\gamma}, \boldsymbol{\beta})\rangle = \prod_{j=1}^{p} e^{-i\beta_j H_x} e^{-i\gamma_j H_z} |+\rangle^{\otimes n}, \tag{1}$$

where $\boldsymbol{\gamma} = (\gamma_1, \gamma_2, \ldots, \gamma_p)$ and $\boldsymbol{\beta} = (\beta_1, \beta_2, \ldots, \beta_p)$ are the $2p$ variational parameters. $|+\rangle^{\bigotimes n}$ corresponds to n qubits in the ground state of $H_x = \sum_{j=1}^{n} X_j$, where X_j is the Pauli X operator acting on the jth qubit.

The symbol p denotes the circuit depth of QAOA. It is defined as how many pairs of operators (layers) are in the quantum circuit. In our context, we use the term *layer*, or usually l, to represent which layer of the circuit is concerned. The total number of layers is equal to the circuit depth. We also use the subscript p to represent a quantity at the circuit depth p, e.g., $|\psi_p\rangle$ means the ansatz state $|\psi\rangle$ at circuit depth p.

Definition 1 (Depth of QAOA circuit). The depth of a QAOA circuit is defined by the number of alternating operator pairs (the cost and mixer Hamiltonians) applied to the initial state, usually denoted as p. A system of the standard QAOA consists of $2p$ parameters.

Definition 2 (Layer of QAOA circuit). The layer of a QAOA circuit is defined as the index of the depth, starting from 1: $l = 1, 2, ..., p$. It is used to represent which layer of the circuit is concerned.

In this paper, we consider the infamous max-cut problem, which maximizes the number of edges between two partitions of a graph. The cost Hamiltonian H_z for the max-cut problem for an unweighted graph $G = (V, E)$ is given as

$$H_z = \frac{1}{2} \sum_{(j,k)\in E} (\mathbb{1} - Z_j Z_k), \tag{2}$$

where Z_j is the Pauli Z operator acting on the jth qubit. We define the expectation of H_z with respect to the ansatz state in Eq. (1):

$$F(\boldsymbol{\gamma}, \boldsymbol{\beta}) = \langle\psi(\boldsymbol{\gamma}, \boldsymbol{\beta})|H_z|\psi(\boldsymbol{\gamma}, \boldsymbol{\beta})\rangle. \tag{3}$$

We can then use a classical optimizer to search for the maximum expectation and the parameters that maximize it. We also define the *approximation ratio* α as

$$\alpha = \frac{F(\boldsymbol{\gamma}^*, \boldsymbol{\beta}^*)}{C_{\max}}, \tag{4}$$

where the superscript * denotes quasi-optimal parameters returned by the classical optimizer and $C_{\max}$ is the maximum cut value (ground truth) for the graph. The approximation ratio is a typical evaluation metric indicating how near the solution given by QAOA is to the ground truth. $0 \leqslant \alpha \leqslant 1$, with the value of 1 nearer to the true solution.

Theoretically, α approaches 1 as $p \to \infty$. However, due to the exponential increase of the number of local minima at larger p, optimizers tend to converge to an undesired local minima if QAOA is initialized randomly. This leads to various initialization strategies being proposed to obtain good-quality solutions for QAOA. Most of the initialization strategies exploit the fact that the optimal parameters of QAOA exhibit a smooth and monotonous pattern, with monotonically increasing γ_i and monotonically decreasing β_i, within the QAOA periodic bound. Depth-progressive initialization strategies, such as the bilinear strategy [23] and INTERP [5], are known to produce high α by fine tuning the parameters depth by depth so that the errors do not accumulate. This is at the cost of depth-progressive strategies requiring the optimization of every p' for $p' = 1, 2, ..., p$, which is very costly. A strategy like TQA [18] uses direct initialization at depth p without the requirement of starting from small depths, but the result is also suboptimal at large depths [17].

Another type of strategy is the optimization strategy, in which the manipulation of parameters occurs during the optimization step. The layerwise strategy [21] falls under this category. The layerwise strategy only updates the parameters of the latest depth. This form of training method greatly reduces the optimization (classical computational) cost, as only two variables need to be optimized, compared with the full optimization of $2p$ variables. However, the layerwise strategy encounters premature saturation in the approximation ratio α, due to the restricted search space with only two latest variables. This behavior of premature saturation is observed in the projector Hamiltonian [21] and the max-cut Hamiltonian [23]. Here, we want to make a clear distinction between the initialization strategies and the optimization strategies as two different categories, for both of them can be combined to work together, or they can work independently on their own.

III. ITERATIVE LAYERWISE

We propose the ITLW strategy, which iteratively updates the QAOA parameters in the layerwise manner, for a certain number of iterations k. We use the notation ITLW(k, p) to denote the ITLW strategy with k iterations for circuit depth p. We can say that the layerwise training is similar to a special case of ITLW at $k = 1$, i.e., layerwise for circuit depth p is similar to ITLW$(1, p)$. However, there is a subtle difference between the layerwise training discussed in Ref. [21] and ITLW$(1, p)$. In layerwise, the parameters are optimized on a shallower circuit starting from $p = 1$, then the QAOA layers are gradually appended as the depth increases. For ITLW$(1, p)$, we first generate $2p$ random parameters for a desired depth p, then the parameters for each layer, (γ_l, β_l) for $l = 1, 2, ..., p$, are optimized separately. This modification is done for a simpler repetition procedure at larger k. Table I shows an example of both strategies for the target depth $p = 3$.

TABLE I. Difference between the layerwise strategy and ITLW(1, p). Layerwise starts with shallower circuits, then more layers are appended as the depth increases. For ITLW, the random full parameter vector of a desired depth p is generated, then they are optimized layer by layer.

Layer	Layerwise		ITLW(1, p)	
	Optimized	Fixed	Optimized	Fixed
$l = 1$	(γ_1, β_1)	–	(γ_1, β_1)	$(\gamma_2, \gamma_3, \beta_2, \beta_3)$
$l = 2$	(γ_2, β_2)	(γ_1, β_1)	(γ_2, β_2)	$(\gamma_1, \gamma_3, \beta_1, \beta_3)$
$l = 3$	(γ_3, β_3)	$(\gamma_1, \beta_1, \gamma_2, \beta_2)$	(γ_3, β_3)	$(\gamma_1, \gamma_2, \beta_1, \beta_2)$

A brief description for the iterative layerwise strategy is as follows. First, choose a QAOA circuit depth p and the number of iterations k. Then, initialize a parameter buffer $\mathbf{\Phi}$ with random parameters of length $2p$: $\mathbf{\Phi} := (\gamma_1, ..., \gamma_p, \beta_1, ..., \beta_p)$. For each iteration, the parameters are optimized in a layerwise manner: each pair of parameters (γ_l, β_l) are optimized layer by layer and update $\mathbf{\Phi}$ with the optimized parameters (γ_l^*, β_l^*), for $l = 1, 2, ..., p$. This procedure is repeated for k iterations. Note that the parameter buffer $\mathbf{\Phi}$ is inherited to the next iteration. The algorithm for ITLW(k, p) is summarized in Algorithm 1.

ALGORITHM 1. Iterative layerwise.

```
Input: No. of iterations $k$, circuit depth $p$.
1: Initialize parameter buffer $\mathbf{\Phi} := (\gamma_1, ..., \gamma_p, \beta_1, ..., \beta_p)$.
2: for $k' = 1, ..., k$ do
3:     for $l = 1, ..., p$ do
4:         $(\gamma_l^*, \beta_l^*) := \arg\max_{\gamma_l, \beta_l} F(\mathbf{\Phi})$
5:         Update $\mathbf{\Phi}$ with $(\gamma_l^*, \beta_l^*)$.
6:     end for
7: end for
8: $\mathbf{\Phi}_p^* := \mathbf{\Phi}$
9: Output: Optimized parameters $\mathbf{\Phi}_p^*$ and their expectation $F_p(\mathbf{\Phi}^*)$.
```

Figure 1 shows the variation of the approximation ratio α with the iteration index k' of ITLW(5, 5), for a graph with six vertices, for ten random sets of initial parameters, i.e., without initialization. Each line represents the variation for one set of initial parameters. It can be seen that the trainability (whether α's improve through the iterations) depends strongly on the initial parameters used to initialize ITLW. Some of the parameters improve greatly through layerwise iterations, while some barely increase after the iterations. This, again, leads to the requirement for a good selection of initial parameters for ITLW.

A. Combining with initialization strategies

We expect a better performance for ITLW using good initializations. In order to generate better initial parameters, we use two types of initialization strategies along with ITLW: the bilinear strategy [23] (depth progressive) and the TQA strategy [18] (direct initialization). The details of the strategies are described in Appendix A, but we will introduce them briefly here.

The bilinear strategy is a depth-progressive strategy which generates the initial parameters for the current depth p with the optimized parameters of the two previous depths: $p-1$ and $p-2$. The strategy is motivated by the linear-like pattern (a smooth, monotonous variation) that QAOA parameters exhibit in two directions: the parameter index and the circuit depth. The optimal parameters in $p-1$ and $p-2$ are extrapolated by taking linear differences to generate the new initial parameters for p. Therefore, optimization is required at every depth in order to find their respective optimal parameters.

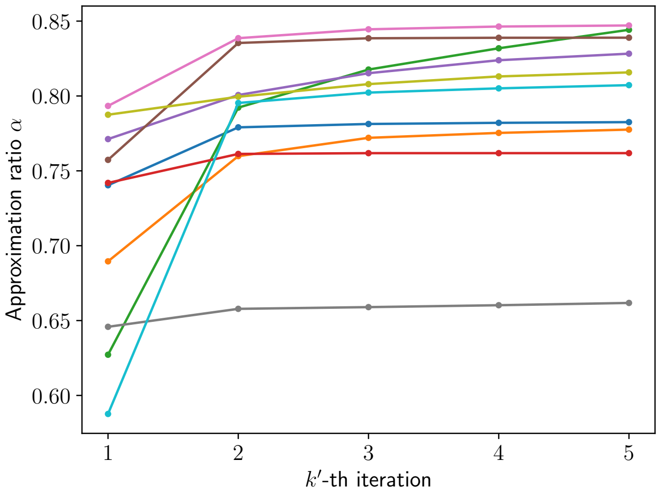

FIG. 1. ITLW(5, 5) applied on solving the QAOA of a six-vertices graph. The lines represent ten different sets of random initial parameters starting at $k' = 1$. The trainability varies depending on the initial parameters. Some α's can be improved through layerwise iterations, while some of them stay almost the same throughout the iterations.

The TQA strategy offers a more direct way to initialize QAOA, without requiring the information of the parameters from previous depths. TQA starts by establishing a linear relation between the parameters (γ_i and β_i) and the total annealing time T. Then, the gradient T of the straight line passing through the origin is optimized to find the straight line which gives the minimum or maximum expectation. QAOA is then initialized with the straight line with the optimized gradient T^*.

The procedure of ITLW combined with initialization strategies is essentially the same as in Algorithm 1. The initialization step is replaced with their respective initialization procedures instead of random initialization. In short, Line 1 of Algorithm 1 is replaced with the initial parameters output from Algorithm 2 (bilinear) or Algorithm IV (TQA) shown in Appendix A. For depth-progressive strategies, an outer loop of $p' = 1, 2, ..., p$ is added (Algorithm 3).

B. Optimization cost

There are many evaluation metrics for the optimization cost of a classical optimizer. We use the number of function evaluations of the expectation function Eq. (3) (number of calls to the quantum circuit) as our evaluation metric for the optimization cost, as this quantity scales with the number of parameters to be optimized. Also, in classical simulations, the optimization time is mostly determined by the number of function evaluations.

Assume that the number of function calls scales with the number of parameters, and hence the circuit depth p. Let $V(p)$ be the number of function calls required to optimize the expectation function F_p given in Eq. (3), for the entire parameter space with $2p$ parameters. We call this the full optimization (FO). For a certain fixed p, it requires $V(p)$ evaluations for FO, while it requires $kpV(1)$ evaluations for ITLW(k, p). This is because for ITLW(k, p), two parameters are optimized per layer for a total of p layers, and for k iterations. Therefore, it is expected that at large p, ITLW will have an advantage in cost over the FO if $V(p)$ scales superlinearly with p, i.e., $V(p) = \mathcal{O}(p^m)$ for $m > 1$, as $kpV(1) = \mathcal{O}(p)$, given that k is a sufficiently small constant.

When used with depth-progressive initialization strategies [5,23], FO requires a total of $S(p) \equiv \sum_{p'=1}^{p} V(p')$ evaluations. This leads to a total cost of $S(p) = \mathcal{O}(p^{m+1})$, given that $V(p) = \mathcal{O}(p^m)$. On the other hand, ITLW calls for a total of $\sum_{p'=1}^{p} kp'V(1) = \mathcal{O}(p^2)$ evaluations. This shows ITLW is also polynomially faster than FO at large p when combined with depth-progressive initialization, under the assumption that $V(p)$ is superlinear. This inference is important for future references and is therefore summarized as Corollary 1.

Corollary 1. At large circuit depth p, ITLW is polynomially faster than FO if $V(p)$ is superlinear with respect to p [$V(p) = \mathcal{O}(p^m)$ for $m > 1$].

IV. RESULTS AND DISCUSSION

We apply the iterative layerwise strategy ITLW(k, p) with different initialization strategies in solving the max-cut problem using QAOA. Our experimental settings are as follows:

(1) Simulator: The quantum circuits are simulated using QISKIT state vector simulation (exact simulation of quantum circuits).

(2) Dataset: We use 30 nonisomorphic graph instances with number of vertices $n = 10$ to $n = 12$. The graph instances are generated randomly, comprising different classes including the three- and four-regular graphs and the Erdös-Rényi graphs with varied edge probabilities.

(3) Initialization strategies: Two different initialization strategies are used: the bilinear strategy [23] and the TQA strategy [18].

(4) Optimization bounds: The parameters are bounded by $\boldsymbol{\gamma} \in [0, \pi)^p$ and $\boldsymbol{\beta} \in [0, \pi/2)^p$ due to the symmetry and the periodicity of QAOA operators [5,17,23,24], so that the optimal parameters reproduce the linear-like pattern for good initialization.

(5) Optimizers: The performances are compared with two different classical optimizers that are widely used in other works: the limited-memory Broyden-Fletcher-Goldfarb-Shanno bounded (L-BFGS-B) algorithm [25] and the Nelder-Mead algorithm [26]. L-BFGS-B is gradient-based and Nelder-Mead is gradient-free. The hyperparameters for the optimizers are the default values provided by the SCIPY package.

(6) Evaluation metrics: The performance of the strategy is evaluated in terms of two different metrics: the approximation ratio α (for the quality of the solution) and the number of function evaluations (for the optimization cost).

(7) No. of ITLW iterations: The performances of different numbers of iterations for ITLW are compared: $k = 1, 2, 3, 4, 5$. We also consider adaptive values for k that depend on the circuit depth p, i.e., $k = \lfloor p/2 \rfloor$ and $k = \lfloor p/2 \rfloor - 1$, where $\lfloor \cdot \rfloor$ represents the floor function.

The main comparison is done between the results produced by ITLW and the FO. We define the *error in the approximation ratio* ε as

$$\varepsilon = \alpha_{\mathrm{FO}} - \alpha_{\mathrm{ITLW}}, \tag{5}$$

where α_{FO} is the approximation ratio obtained from FO, and α_{ITLW} is the approximation ratio from ITLW. Note that to write Eq. (5) out explicitly, it will become

$$\varepsilon_p = \alpha_{p,\mathrm{FO}} - \alpha_{p,\mathrm{ITLW}}. \tag{6}$$

However, for simplicity, we will omit the subscript p as we assume that the readers are aware that this quantity (and those defined later) are different at different p's. The quantity ε allows us to know how much the α given by ITLW deviates from that of FO. If $\varepsilon > 0$, FO approximates better, and if $\varepsilon < 0$, ITLW approximates better. Note that since $0 \leqslant \alpha \leqslant 1$, ε can take values between the range $-1 \leqslant \varepsilon \leqslant 1$.

We define another quantity, known as the *cost reduction ratio* r, to evaluate how much reduction in the number of function evaluations for ITLW:

$$r = \frac{\tilde{V}_{\mathrm{ITLW}}}{\tilde{V}_{\mathrm{FO}}}. \tag{7}$$

We use the notation $\tilde{V}$ to denote the empirical number of function evaluations (instead of the theoretical V in the previous section). If $r < 1$, it means there is a reduction in the number of function evaluations for ITLW compared to FO. If $r > 1$, it means ITLW costs more than FO.

For depth-progressive strategies, we introduce another quantity called the *cumulative-cost reduction ratio* r_c:

$$r_c = \frac{\sum_p \tilde{V}_{p,\mathrm{ITLW}}}{\sum_p \tilde{V}_{p,\mathrm{FO}}}, \tag{8}$$

where $\sum_p \tilde{V}_p$ means the sum of the number of function evaluations up to the target depth p. This is a more objective quantity for the total cost as we need to consider the cost for every depth for depth-progressive strategies.

When calculating ε, r, and r_c, we use the values that are generated under the same experimental conditions, e.g., same optimizer, same circuit depth, and same initialization strategy, so that the only difference between the values is caused by the different optimization strategies, i.e., ITLW and FO. However, since FO does not have the parameter k, we use the same value for all k's when comparing with ITLW, while keeping the other conditions the same. For a simpler interpretation, having lower values of the quantities ε and r (and r_c) implies better results.

A. Bilinear initialization

Figure 2 shows the results of ITLW with bilinear initialization, averaged over the 30 graphs that we considered. For the

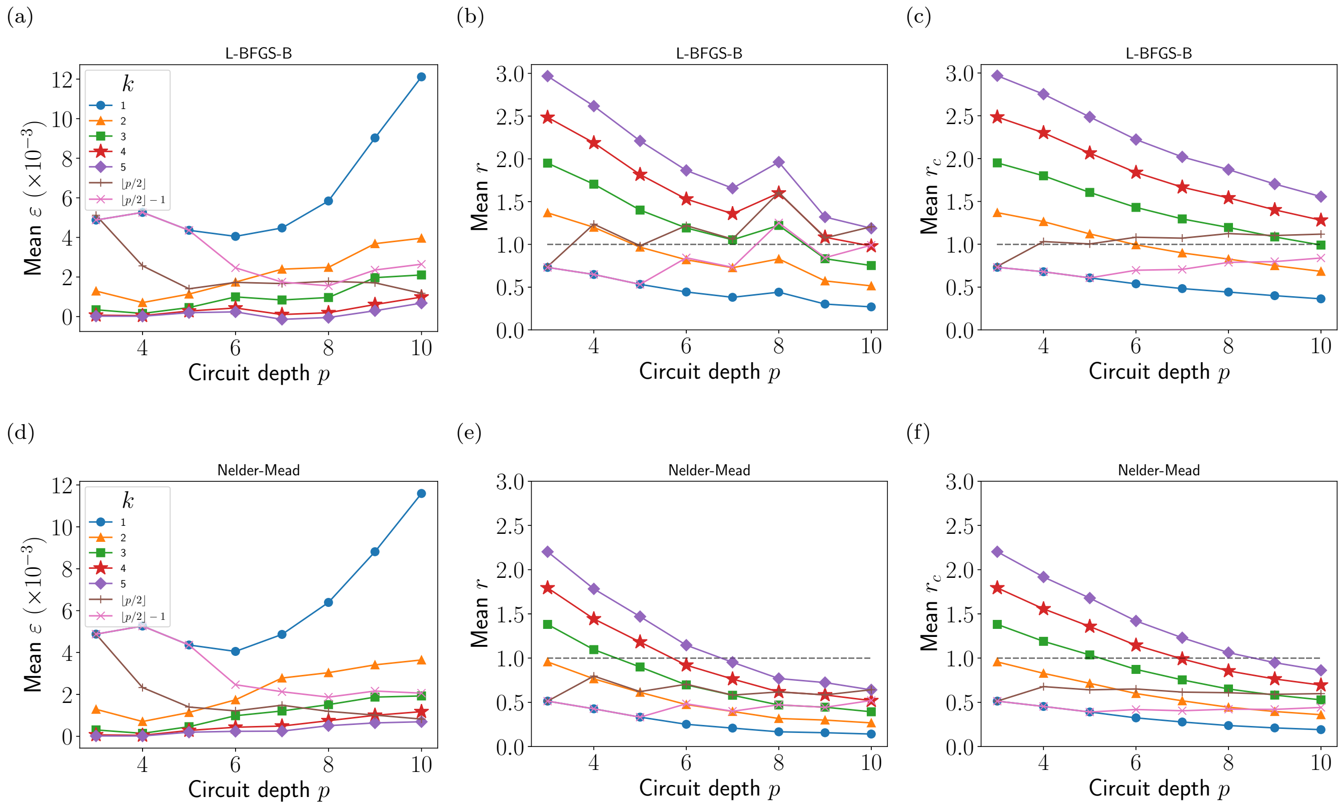

FIG. 2. The results for ITLW with bilinear initialization. The columns show the mean error in approximation ratio ε, the cost reduction ratio r, and the cumulative-cost reduction ratio r_c, respectively. Different colors of lines show the results for different number of iterations k. The top row shows the results for the L-BFGS-B optimizer. The bottom row shows the results for the Nelder-Mead optimizer. The dashed gray lines in the r and r_c figures show the critical ratio of 1 when ITLW has an advantage over FO.

bilinear initialization, we initialize from $p = 3$, then slowly raise the depth to $p = 10$. Each depth is optimized using the ITLW strategy. Figures 2(a)–2(c) show the results for the L-BFGS-B optimizer; Figs. 2(d)–2(f) show the results for the Nelder-Mead optimizer. It can be observed that the overall trend does not change much when comparing Figs. 2(a) and 2(d), implying that the choice of optimizers does not have much effect on the errors in α under this experimental condition. Figure 2(a) shows the errors ε decrease as the number of iterations k increases. This says that the parameters generated from the bilinear strategy are trainable as the error decreases with increasing k (α_{FO} is the same for all k for a certain p). Also, the strategy is able to achieve a considerably small error $\varepsilon < 4 \times 10^{-3}$ even for a small k at ITLW(2, 10) [the orange triangle markers in Fig. 2(a)]. The error ε continues to decrease as k increases.

Figures 2(b) and 2(e) show the cost reduction ratio r averaged over 30 graph instances. The horizontal gray dashed lines show the critical value $r = 1$, in which ITLW costs less (faster) than FO if r is below the critical value. Overall, the Nelder-Mead optimizer has lower r compared to the L-BFGS-B optimizer. The general trend shows that r is lower for larger circuit depth p, caused by the difference between the costs of ITLW and FO mentioned in Sec. III B. This result agrees with Corollary 1. Recall that the costs for ITLW and FO differ by a polynomial degree of p, so we would expect that more cost is saved at larger p. For L-BFGS-B, ITLW(4, 10) is able to achieve a borderline advantage at $r \approx 1$ [the red star markers in Fig. 2(b)]. Although ITLW(5, 10) fails to achieve an advantage, it is expected to have an even lower r if p is further increased. For Nelder-Mead, the cost reduction ratio is lower than 1 for all k at the target depth $p = 10$. Although the cost is lower at larger circuit depths, the errors have a tendency to increase at larger p as shown in Figs. 2(a) and 2(d). Thus, the ITLW strategy is more like a trade-off strategy rather than a strategy with an absolute advantage. The approximation ratio is compromised in exchange for a reduction in cost. However, we still need to consider whether it is worth it to exchange a reduction in α by 4×10^{-3} given that the cost is almost halved [in the case of ITLW(2, 10)].

For the adaptive number of iterations $k = \lfloor p/2 \rfloor$, the mean error at $p = 10$ is near to that of $k = 4$ for L-BFGS-B, and is near to the mean error of $k = 5$ for Nelder-Mead. It can be observed from Figs. 2(c) and 2(f) that the total cost for the ITLW is less than that of FO for this adaptive k (comparing the red star and the brown plus markers), which means it is worth switching to the adaptive strategy. However, since the cost for the constant k will decrease further for larger p, we expect that the adaptive k will cost more than the constant k at larger p. For $k = \lfloor p/2 \rfloor - 1$, the same patterns occur. The mean error at $p = 10$ is near to that of $k = 3$ and the cost is less. The cost is expected to grow for larger p. It is interesting to compare both adaptive values for the L-BFGS-B optimizer, where the total cost of ITLW is less than that of FO for $\lfloor p/2 \rfloor - 1$, but is more than FO for $\lfloor p/2 \rfloor$, despite only having one iteration difference for each depth.

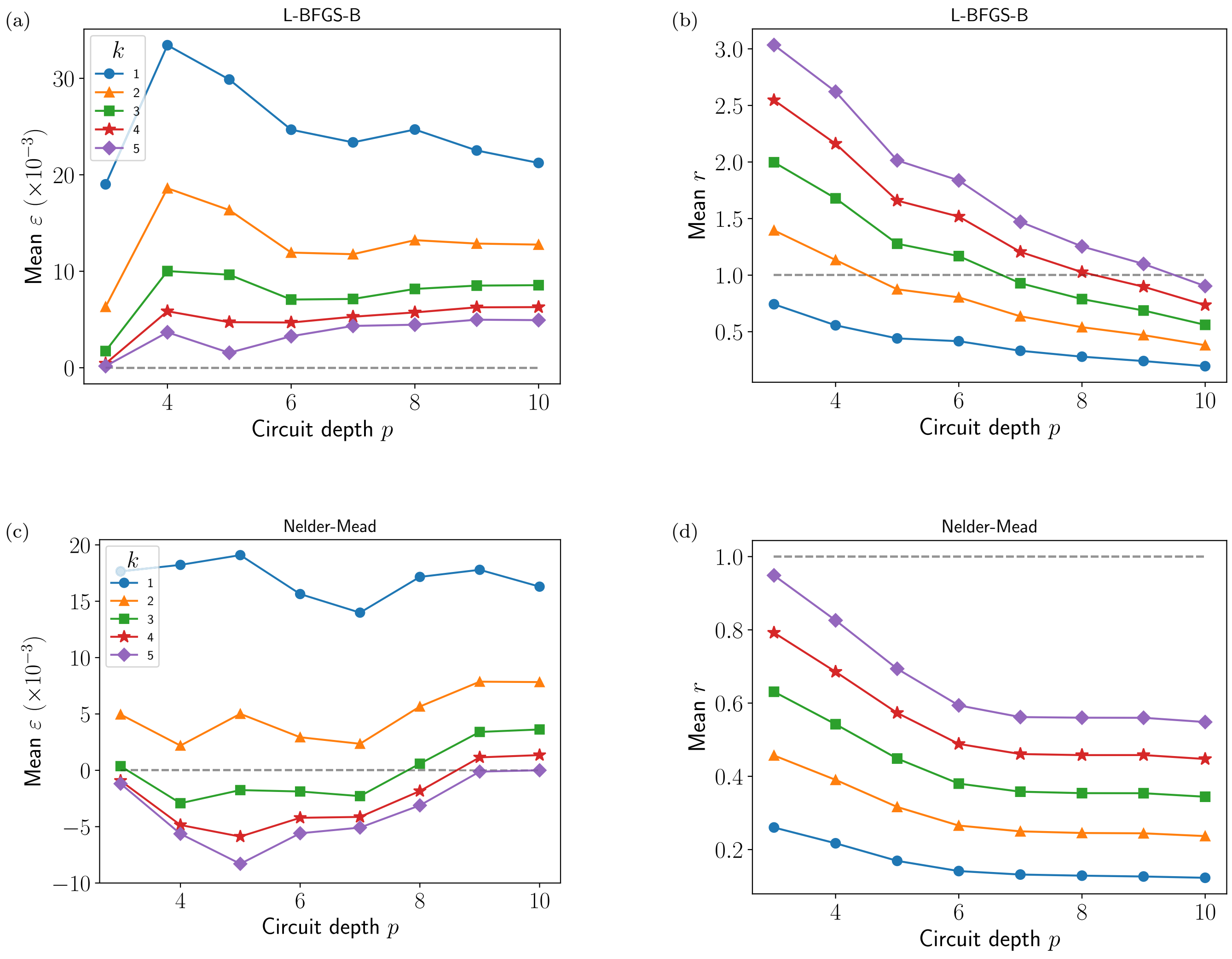

FIG. 3. The results for ITLW with TQA initialization. The mean error in approximation ratio ε and the cost reduction ratio r, respectively. The top row shows the results for the L-BFGS-B optimizer and the bottom row shows the results for the Nelder-Mead optimizer. The dashed gray lines show the critical values $\varepsilon = 0$ and $r = 1$ when ITLW has an advantage over FO.

B. TQA initialization

Figure 3 shows the results for ITLW with the TQA initialization, relative to FO. The general trend for ε [shown in Figs. 3(a) and 3(c)] with increasing k remains as expected. As k increases, the overall mean ε decreases, which shows the approximation ratio α improves as we increase the number of iteration. The Nelder-Mead optimizer also gives less errors than the L-BFGS-B optimizer for the same k. Also, for the Nelder-Mead optimizer, as k increases, we can see that ITLW yields higher α than FO, which is rarely seen in the ITLW with bilinear initialization. The improvement is as much as 8×10^{-3} at ITLW(5, 5) (purple diamond marker). This might be because the α obtained from TQA is relatively lower than that obtained from the bilinear, giving more room for improvement when ITLW is applied.

Figures 3(b) and 3(d) show the respective cost reduction ratio for L-BFGS-B and Nelder-Mead. For both of the optimizers, the mean r decreases monotonically as p increases. Comparing the cost reduction ratio of the two optimizers, we can see a large difference in the magnitude of the values of r. However, for L-BFGS-B, the costs still pass through the critical value $r = 1$ as p increases. For Nelder-Mead, $r < 1$ for all k's, despite having higher α than FO for $k = 3, 4, 5$ at some of the depths. This shows that the Nelder-Mead optimizer exhibits a significant cost advantage with TQA initialization.

Since TQA is a direct initialization method, we did not include the results (ε and r) for the adaptive k as they are the same as those at their respective number of iterations.

V. CONCLUSIONS

Our research brought out the possibility of combining initialization strategies and optimization strategies in QAOA. We proposed the ITLW optimization strategy for QAOA and use it to solve max-cut problems with the size of 10–12 qubits. As an improvement to the layerwise training, ITLW prevents the premature saturation that occurs in the layerwise training, which allows the parameters to be further trained to achieve a higher approximation ratio. However, the trainability of the parameters is still highly sensitive to the initial parameters, so we combine ITLW with the bilinear and TQA initialization strategies for a better initialization. Instead of being a strategy with an absolute advantage, the strategy imposes a trade-off between the approximation ratio and the optimization cost. The simulation results show that ITLW has a lower optimization cost in exchange for a lower approximation ratio on average, compared to FO. The cost reduction gets more significant as the QAOA circuit depth increases. By combining with the bilinear initialization strategy, ITLW is able to reduce the cost by half in exchange for a reduction of 4×10^{-3} in the approximation ratio [see orange triangles in Fig. 2(a)]. There are some exceptions in the case of the TQA strategy optimized with the Nelder-Mead optimizer, where we observe slight improvements in the approximation ratio at shallow depths, despite the cost reduction [c.f. Fig. 3(c)].

Although we proposed and tested ITLW on QAOA, this idea has a broader view in the sense that it can also be applied in other VQAs, such as the variational quantum eigensolver (VQE) and the variational circuits in quantum machine learning. Some details can also be tweaked—for example, the optimization can be parameter-wise (optimize parameter by parameter) instead of layer-wise to yield a further reduction in the optimization cost.

APPENDIX A: DETAILS OF THE INITIALIZATION STRATEGIES

In this section, we briefly discuss the details of the initialization strategies used in conjunction with ITLW. We used two different initialization strategies: the bilinear strategy [23] for depth-progressive (depth-by-depth) initialization, and the TQA strategy for direct initialization.

ALGORITHM 2. Bilinear initialization for a single depth.

```
Input: Circuit depth $p$, $\mathbf{\Phi}^*_{p-1}$ and $\mathbf{\Phi}^*_{p-2}$.
1: for $i = 1...p$ do
2:     if $i \leqslant p-2$ then
3:         $\phi^p_i := \phi^{p-1}_i + \Delta^{p-1,p-2}_i$
4:     else if $i = p-1$ then
5:         $\phi^p_i := \phi^{p-1}_i + \Delta^{p-1,p-2}_{i-1}$
6:     else if $i = p$ then
7:         $\phi^p_i := \phi^p_{i-1} + \Delta^p_{i-1,i-2}$
8:     end if
9: end for
Output: Initial parameters $\mathbf{\Phi}_p$ for QAOA at depth $p$.
```

The bilinear strategy takes the optimal parameters from two previous depths $p-1$ and $p-2$ to predict the initial parameters for the current depth p. It leverages the linear-like pattern of the QAOA optimal parameters discussed in many works [5,27], which resembles the linear annealing schedule. Moreover, it is also discovered that the values of the optimal parameters do not stay the same for different depths, coined as *nonoptimality* in Ref. [23]. Using these properties, the parameters for p can be extended by taking the linear difference between the parameters in $p-1$ and $p-2$. The idea is to extrapolate the amount of difference between $\mathbf{\Phi}^*_{p-1}$ and $\mathbf{\Phi}^*_{p-2}$ to $\mathbf{\Phi}_p$, where $\mathbf{\Phi}_p \equiv (\boldsymbol{\gamma}, \boldsymbol{\beta})_p$ are the parameters at depth p. ϕ^p_i denotes the parameter (regardless of γ or β) at depth p with index i. For parameters with indices $i \leqslant p-2$, the parameters are extrapolated using nonoptimality:

$$\begin{aligned}\phi^p_i &= \phi^{p-1}_i + \Delta^{p-1,p-2}_i \\ &= 2\phi^{p-1}_i - \phi^{p-2}_i,\end{aligned} \tag{A1}$$

where the notation $\Delta_{i,j} \equiv \phi_i - \phi_j$. This also applies to the symbols on the superscript. For the parameters with $i = p-1$, we cannot use Eq. (A1) to extend as the parameter ϕ^{p-2}_{p-1} does not exist ($j \leqslant p$ for ϕ^p_j), so we take the difference from the previous index $i = p-2$ instead:

ALGORITHM 3. Depth-progressive bilinear initialization with ITLW.

```
Input: Target depth $p_t$, $\mathbf{\Phi}^*_1$ and $\mathbf{\Phi}^*_2$.
1: for $p = 3...p_t$
2:     Build the initial parameters $\mathbf{\Phi}_p$:
3:     for $i = 1...p$ do
4:         if $i \leqslant p-2$ then
5:             $\phi^p_i := \phi^{p-1}_i + \Delta^{p-1,p-2}_i$
6:         else if $i = p-1$ then
7:             $\phi^p_i := \phi^{p-1}_i + \Delta^{p-1,p-2}_{i-1}$
8:         else if $i = p$ then
9:             $\phi^p_i := \phi^p_{i-1} + \Delta^p_{i-1,i-2}$
10:        end if
11:    end for
12:    Initialize QAOA with $\mathbf{\Phi}_p$ and perform optimization with
       ITLW (Algorithm 1).
13: end for
Output: Optimal parameters $\mathbf{\Phi}^*_{p_t}$ and $F(\mathbf{\Phi}^*_{p_t})$ for the target depth $p_t$.
```

$$\phi^p_{p-1} = \phi^{p-1}_{p-1} + \Delta^{p-1,p-2}_{p-2}. \tag{A2}$$

For the newly added parameter, we use the linear-like adiabatic pattern to extend

$$\begin{aligned}\phi^p_p &= \phi^p_{p-1} + \Delta^p_{p-1,p-2} \\ &= 2\phi^p_{p-1} - \phi^p_{p-2}.\end{aligned} \tag{A3}$$

Equations (A1)–(A3) are essentially all the rules required to generate the new initial parameters $\mathbf{\Phi}_p$. The overall procedure for the bilinear strategy (and the depth-progressive version starting from $p = 3$) is summarized in Algorithm 2 (Algorithm 3).

ALGORITHM 4. TQA initialization.

```
Input: Circuit depth $p$.
1: Establish the relation between the parameters and the
   annealing time $T$, and hence $F(T)$:
   $\gamma_i(T) = iT/p$;
   $\beta_i(T) = (1 - i/p)T/p$;
   $\langle H_z \rangle = F(T) = F[\boldsymbol{\gamma}(T), \boldsymbol{\beta}(T)]$.
2: Optimize $F(T)$ to obtain the optimal annealing time $T^*$.
Output: Initial parameters $[\boldsymbol{\gamma}(T^*), \boldsymbol{\beta}(T^*)]$.
```

The TQA strategy aims to find the "optimal annealing time" T for the QAOA to generate the initial parameters, using the angle-index relation: $\gamma_i = i\Delta t/p$; $\beta_i = (1 - i/p)\Delta t$, where i is the angle index. Δt is the *annealing time step* and can be related with the *total annealing time* T with the relation $\Delta t = T/p$. Hence, we get the angle-index relation in terms

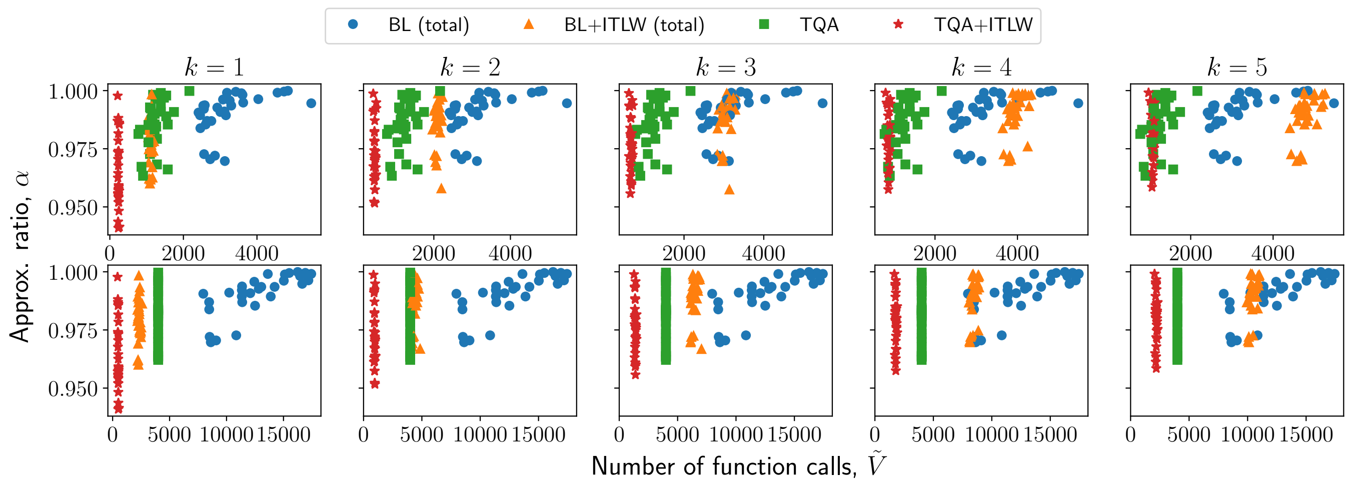

FIG. 4. The approximation ratio α vs the number of function calls $\tilde{V}$ for all the graph instances at depth $p = 10$. The top row shows the results for the L-BFGS-B optimizer. The bottom row shows the results for the Nelder-Mead optimizer. Each column shows a different number of iteration k. Each plot shows four methods: the bilinear strategy (BL), bilinear combined with iterative layerwise (BL + ITLW), TQA, and TQA combined with iterative layerwise (TQA + ITLW).

of T:

$$\gamma_i(T) = \frac{iT}{p^2}; \quad \beta_i(T) = \left(1 - \frac{i}{p}\right)\frac{T}{p}. \tag{A4}$$

Substituting Eq. (A4) into (3) gives the expectation function in terms of T. Then, the optimal annealing time T^* is found by maximizing $F(T)$:

$$T^* = \arg\max_T F(T). \tag{A5}$$

T^* is then substituted back into (A4) to generate the initial angles $[\boldsymbol{\gamma}(T^*), \boldsymbol{\beta}(T^*)]$. The initial angles generated take a linear form against the angle index i. We summarized the procedure for TQA in Algorithm 4.

APPENDIX B: OVERALL RESULTS

Figure 4 shows the exact values of the approximation ratio α versus number of function calls $\tilde{V}$, instead of the relative values shown in Figs. 2 and 3. The data shown is at circuit depth $p = 10$.

[1] E. Farhi, J. Goldstone, and S. Gutmann, A quantum approximate optimization algorithm, arXiv:1411.4028 [quant-ph].

[2] E. Farhi, J. Goldstone, S. Gutmann, and M. Sipser, Quantum computation by adiabatic evolution, arXiv:quant-ph/0001106.

[3] S. Lloyd, Quantum approximate optimization is computationally universal, arXiv:1812.11075 [quant-ph].

[4] G. Crooks, Performance of the quantum approximate optimization algorithm on the maximum cut problem, arXiv:1811.08419.

[5] L. Zhou, S.-T. Wang, S. Choi, H. Pichler, and M. D. Lukin, Quantum approximate optimization algorithm: Performance, mechanism, and implementation on near-term devices, Phys. Rev. X **10**, 021067 (2020).

[6] C. Moussa, H. Calandra, and V. Dunjko, To quantum or not to quantum: Towards algorithm selection in near-term quantum optimization, Quantum Sci. Technol. **5**, 044009 (2020).

[7] K. Marwaha, Local classical MAX-CUT algorithm outperforms $p = 2$ QAOA on high-girth regular graphs, Quantum **5**, 437 (2021).

[8] S. Hadfield, Z. Wang, B. O'Gorman, E. Rieffel, D. Venturelli, and R. Biswas, From the quantum approximate optimization algorithm to a quantum alternating operator ansatz, Algorithms **12**, 34 (2019).

[9] L. Zhu, H. L. Tang, G. S. Barron, F. A. Calderon-Vargas, N. J. Mayhall, E. Barnes, and S. E. Economou, Adaptive quantum approximate optimization algorithm for solving combinatorial problems on a quantum computer, Phys. Rev. Res. **4**, 033029 (2022).

[10] A. Bartschi and S. Eidenbenz, Grover mixers for QAOA: Shifting complexity from mixer design to state preparation, in *2020 IEEE International Conference on Quantum Computing and Engineering (QCE)* (IEEE, Piscataway, NJ, 2020).

[11] R. Herrman, P. C. Lotshaw, J. Ostrowski, T. S. Humble, and G. Siopsis, Multi-angle quantum approximate optimization algorithm, arXiv:2109.11455 [quant-ph].

[12] E. Grant, L. Wossnig, M. Ostaszewski, and M. Benedetti, An initialization strategy for addressing barren plateaus in parametrized quantum circuits, Quantum **3**, 214 (2019).

[13] A. V. Uvarov and J. D. Biamonte, On barren plateaus and cost function locality in variational quantum algorithms, J. Phys. A: Math. Theor. **54**, 245301 (2021).

[14] M. Cerezo, A. Sone, T. Volkoff, L. Cincio, and P. J. Coles, Cost function dependent barren plateaus in shallow parametrized quantum circuits, Nat. Commun. **12**, 1791 (2021).

[15] S. Wang, E. Fontana, M. Cerezo, K. Sharma, A. Sone, L. Cincio, and P. J. Coles, Noise-induced barren plateaus in variational quantum algorithms, Nat. Commun. **12**, 6961 (2021).

[16] G. G. Guerreschi and A. Y. Matsuura, Qaoa for max-cut requires hundreds of qubits for quantum speed-up, Sci. Rep. **9**, 6903 (2019).

[17] S. H. Sack, R. A. Medina, R. Kueng, and M. Serbyn, Recursive greedy initialization of the quantum approximate optimization algorithm with guaranteed improvement, Phys. Rev. A **107**, 062404 (2023).

[18] S. H. Sack and M. Serbyn, Quantum annealing initialization of the quantum approximate optimization algorithm, Quantum **5**, 491 (2021).

[19] X. Lee, Y. Saito, D. Cai, and N. Asai, Parameters fixing strategy for quantum approximate optimization algorithm, in *2021 IEEE International Conference on Quantum Computing and Engineering (QCE)* (IEEE, Piscataway, NJ, 2021).

[20] R. Shaydulin, I. Safro, and J. Larson, Multistart methods for quantum approximate optimization, in *2019 IEEE High Performance Extreme Computing Conference (HPEC), Waltham,MA* (IEEE, Piscataway, NJ, 2019), pp. 1–8.

[21] E. Campos, D. Rabinovich, V. Akshay, and J. Biamonte, Training saturation in layerwise quantum approximate optimization, Phys. Rev. A **104**, L030401 (2021).

[22] A. Skolik, J. R. McClean, M. Mohseni, P. van der Smagt, and M. Leib, Layerwise learning for quantum neural networks, Quantum Mach. Intell. **3**, 5 (2021).

[23] X. Lee, N. Xie, D. Cai, Y. Saito, and N. Asai, A depth-progressive initialization strategy for quantum approximate optimization algorithm, Mathematics **11**, 2176 (2023).

[24] P. C. Lotshaw, T. S. Humble, R. Herrman, J. Ostrowski, and G. Siopsis, Empirical performance bounds for quantum approximate optimization, Quantum Info. Proc. **20**, 403 (2021).

[25] J. L. Morales and J. Nocedal, Remark on "Algorithm 778: L-bfgs-b: Fortran subroutines for large-scale bound constrained optimization," ACM Trans. Math. Softw. **38**, 1 (2011).

[26] J. A. Nelder and R. Mead, A simplex method for function minimization, Comput. J. **7**, 308 (1965).

[27] J. Cook, S. Eidenbenz, and A. Bärtschi, The quantum alternating operator ansatz on maximum k-vertex cover, in *2020 IEEE International Conference on Quantum Computing and Engineering (QCE), Denver, CO* (IEEE, Piscataway, NJ, 2020), p. 83.