

Extrapolation method to optimize linear-ramp quantum approximate optimization algorithm parameters: Evaluation of runtime scaling

Vanessa Dehn¹, Martin Zaefferer², Gerhard Hellstern³, Karthik Jayadevan,¹
 Florentin Reiter¹ and Thomas Wellens^{1,*}

¹*Fraunhofer Institute for Applied Solid State Physics IAF, Tullastr. 72, 79108 Freiburg, Germany*

²*DHBW Ravensburg, Marienplatz 2, 88212 Ravensburg, Germany*

³*DHBW Stuttgart, Herdweg 20, 70174 Stuttgart, Germany*



(Received 17 April 2025; accepted 15 January 2026; published 9 March 2026)

The quantum approximate optimization algorithm (QAOA) has been suggested as a promising candidate for the solution of combinatorial optimization problems. Yet, whether—or under what conditions—it may offer an advantage compared with classical algorithms remains to be proven. Using the standard variational form of QAOA requires a high number of circuit parameters that have to be optimized at a sufficiently large depth, which constitutes a bottleneck for achieving a potential scaling advantage. The linear-ramp QAOA has been proposed to address this issue, as it relies on only two parameters that have to be optimized. Based on this, we develop a method to estimate suitable values for those parameters through extrapolation, starting from smaller problem sizes (number of qubits) toward larger problem sizes. We apply this method to several use cases, such as portfolio optimization, feature selection, clustering, and weighted maxcut. From results obtained on a noiseless quantum emulator, we evaluate the quantum runtime scaling for finding the optimal solution and compare it with that of classical methods. In the case of portfolio optimization, we demonstrate superior scaling compared with the classical runtime for the problem sizes of up to 28 qubits that we consider in this work.

DOI: [10.1103/PhysRevA.113.032413](https://doi.org/10.1103/PhysRevA.113.032413)

I. INTRODUCTION

The quantum approximate optimization algorithm (QAOA) is a widely studied and promising algorithm for the solution of combinatorial optimization problems [1–4], potentially more efficient than classical algorithms in terms of faster runtimes (time to find the optimal solution) or finding solutions of higher quality [5–7]. However, the standard variational QAOA ansatz, where the variational parameters of the quantum circuits are optimized in connection with a classical optimization routine, is not expected to lead to such a possible runtime advantage. This can be understood from the complexity of optimizing these parameters as follows: the quantum circuit consists of $2p$ circuit parameters $(\vec{\gamma}, \vec{\beta}) = (\gamma_1, \dots, \gamma_p, \beta_1, \dots, \beta_p)$ that, in standard QAOA, all need to be optimized for a sufficiently large depth p . This requires a high number of circuit evaluations. Additionally, each evaluation itself contains a substantial number of shots (typically around 1000 shots or more) to precisely determine the expectation value of the cost function, which is minimized by the variational algorithm. Furthermore, when using the variational ansatz, the classical optimizer can get stuck in local minima when optimizing $\vec{\gamma}$ and $\vec{\beta}$ [8,9], especially

without a good initial guess. Therefore, fixed schedules, i.e., structured and predefined approaches for selecting circuit parameters, are of interest to circumvent expensive optimization routines [6,9,10]. It has been observed that the optimal parameters $(\vec{\gamma}^*, \vec{\beta}^*)$ exhibit patterns, namely, that γ_j^* increases smoothly while β_j^* shows a decreasing behavior (with $j = 1, \dots, p$) [11]. These patterns are consistent with QAOA being interpreted as realizing a discrete adiabatic schedule with gradually changing unitaries [8].

Based on these patterns, schedules can be formulated by which circuit parameters $(\vec{\gamma}, \vec{\beta})$ are determined for, in principle, arbitrarily high depth p . Examples for such patterns are the Fourier heuristic strategy [11], Trotterized quantum annealing (TQA) [12], or the linear-ramp schedule [8,13–15], among others [16,17]. Sometimes (e.g., in Refs. [11,12]), such schedules are used in order to provide initial parameters, which are then further optimized according to standard QAOA. Other works [8,13–15] propose to use these schedules as fixed schedules, thus saving the enormous overhead involved in further optimizing a large number of circuit parameters, as explained above.

Indeed, a recent study claims that fixed linear-ramp schedules provide a runtime scaling advantage for some combinatorial optimization problems (in particular for a fully connected, randomly weighted MaxCut problem) [18]. Apart from the QAOA depth p , the linear-ramp schedule exhibits only two free parameters Δ_γ and Δ_β (see Sec. IV A below). The authors of Ref. [18] suggest that the parameter optimization needs to be performed only for one single problem instance of given size N , and then the same parameters can be used also for other instances of the same size. Nevertheless,

*Contact author: thomas.wellens@iaf.fraunhofer.de

Published by the American Physical Society under the terms of the Creative Commons Attribution 4.0 International license. Further distribution of this work must maintain attribution to the author(s) and the published article's title, journal citation, and DOI.

finding the optimal parameters even for a single problem instance at large N constitutes a considerable effort. The question remains how suitable values of the parameters Δ_γ , Δ_β and p can be found in an efficient and scalable way. In our work, we address this question by proposing an extrapolation scheme whereby circuit parameters for larger problem sizes (in our case, up to $N = 28$) can be determined on the basis of optimizations performed only on subproblems of smaller size ($N = 4, 6, 8, 10$). With these parameters, we present the scaling of the quantum runtime for four combinatorial optimization problems: portfolio optimization, feature selection, clustering, and weighted MaxCut. The runtime is expressed as the median of the total depth of all quantum circuits that have to be executed to find the optimal solution. After comparing the scaling of the quantum runtime with the classical runtime, we find an indication of a quantum scaling advantage in the case of portfolio optimization.

Apart from the article [18] mentioned above, similar scaling advantages have recently been found also for boolean satisfiability problems [19] and the low autocorrelation binary sequences (LABSs) problem [20]. In both cases, fixed parameter schedules (though not as simple as linear-ramp) are considered, and the scaling of the QAOA runtime (combined with quantum minimum finding in [20]) is analyzed at fixed QAOA depth p . In our work, we propose to adapt both the depth p and the circuit parameters ($\vec{\gamma}$, $\vec{\beta}$) with increasing problem size N . We therefore think it is possible that the QAOA runtime scalings obtained in the above-mentioned works can be further improved when combining them with the ideas presented in this article.

This paper is divided into two main parts, a classical optimization and a quantum optimization part, and is organized as follows: in Sec. II, we give a brief introduction to the combinatorial optimization problems considered in this paper, which are then addressed in Sec. III by purely classical methods. This section provides the investigation as well as the discussion of the scaling behavior of different classical optimization methods (CPLEX, GUROBI, the MQLib/Burer2002 Heuristic, and the Goemans-Williamson algorithm), which is evaluated by the time to find the optimal solution (runtime) as a function of the number of variables (bits). Section IV gives an introduction to the linear-ramp QAOA schedule and an explanation of our method to derive suitable QAOA parameters based on extrapolation from a smaller number of qubits toward a larger number of qubits. The results are then discussed in Sec. V and we finally conclude in Sec. VI.

II. COMBINATORIAL OPTIMIZATION PROBLEMS

In this section, we introduce four optimization problems that we all formulate as a quadratic unconstrained binary optimization (QUBO) [21] problem with the following structure:

$$F(\mathbf{z}) = \sum_{i,j}^N F_{ij} z_i z_j + \sum_i^N f_i z_i, \quad (1)$$

with symmetric matrix F , vector f , number of bits N , and binary variables $z_1, \dots, z_N$.

Taking into account that $z_i z_i = z_i$ (for binary variables $z_i \in \{0, 1\}$) the linear term can be included in the quadratic term,

leading to

$$F(\mathbf{z}) = \sum_{i,j}^N Q_{ij} z_i z_j, \quad (2)$$

where $Q_{ij} = F_{ij} + f_i \delta_{ij}$ is a symmetric matrix defining the problem under consideration. The optimization task is to minimize the cost function $F(\mathbf{z})$, Eq. (2).

Before discussing different combinatorial optimization problems, we want to stress that these examples are simplified compared with real-world problems in business and industry. While it would be possible to make them more realistic, it will be seen in the following that they nevertheless obtain generic features.

A. Formulation of the portfolio problem

The portfolio optimization problem is one of the most famous and investigated topics in finance after Markowitz published his seminal paper [22]. It addresses the question of how to build a portfolio with different financial assets in order to maximize the return and to minimize the risk when the relation between these two properties is fixed. Whereas in banks and asset management firms a lot of subtleties have to be considered (i.e., use of sophisticated risk measures as value-at-risk, allowing for short-selling, use of integer asset fractions, transaction costs, etc.), a generic formulation is obtained within the following framework: Use the historic return as a measure of the expected return, use the covariance of the asset returns as risk measure and restrict yourself to binary portfolio fractions. Using this together with the aforementioned QUBO function, see Eq. (1), we can directly identify the cost function for the portfolio optimization problem,

$$F_{\text{portfolio}}(\mathbf{z}) = q \sum_{i,j}^N z_i z_j \sigma_{ij} - (1 - q) \sum_i^N z_i \mu_i, \quad (3)$$

with number of assets N , covariance matrix σ_{ij} , expected return vector μ_i , binary portfolio weights $z_i = 0$ or 1 of stock i , and risk preference factor q . A budget constraint $B = \sum_i z_i$ indicates the investment of a fixed budget, with B being the number of assets chosen for the portfolio from N total assets. A created portfolio is called feasible if the constraint is fulfilled; otherwise, we call a portfolio infeasible. The constraint can be added to the cost function as a penalty term with a prefactor A :

$$F_{\text{portfolio}}^{(A)}(\mathbf{z}) = F_{\text{portfolio}}(\mathbf{z}) + A \left(\sum_{i=1}^N z_i - B \right)^2. \quad (4)$$

A suitable candidate for the prefactor A can be computed using the routine described in Ref. [23]. In continuation of this paper, we use the annualized returns and covariance matrices for the German DAX 30 (as of 1/2/2021) between 01/01/2016 and 31/12/2020. For a given number of assets, different instances of the problem are obtained by randomly selecting N out of the 30 DAX assets. The risk preference factor q is typically set between 0 and 1 and, in contrast with our typical choice of $q = 1/3$, it is set here to $q = 1$, indicating the focus on the risk part of the problem irrespective of the return part. Concerning the budget, we choose $B = N/2$ (for even N) or $B = (N - 1)/2$ (for odd N).

B. Formulation of feature selection problem

Similarly, we can formulate the cost function for the feature selection problem in machine learning, which can be based on two dependency measures: the first describes the dependence $\rho_{i,j}$ between the columns i and j of the feature matrix X , and the second is the dependence $\rho_{i,Y}$ between the feature column i and the target vector Y . Furthermore, we consider a set of binary variables $z_i \in \{0, 1\}$ for which $z_i = 1$ indicates that the column i is selected in the model, while $z_i = 0$ refers to the column not being selected in the model. An intuitive approach is to select features such that the interfeature dependence $\sum_{i,j=1}^N z_i z_j |\rho_{i,j}|$ is minimized and the dependence between feature and target, $\sum_{i=1}^N z_i |\rho_{i,Y}|$, is maximized. The cost function can now be constructed as above by combining both criteria:

$$F_{\text{feature}}(\mathbf{z}) = - \left(\phi \sum_{i=1}^N z_i |\rho_{i,Y}| - (1 - \phi) \sum_{\substack{i,j=1 \\ i \neq j}}^N z_i z_j |\rho_{i,j}| \right), \quad (5)$$

with weighting parameter $\phi \in [0, 1]$, which we set to $\phi = 0.9$. As a dependence measure between the features as well as between features and target, we chose to take the linear correlation ρ^{Correl} as proposed in Ref. [24]. In accordance with this reference, the “German Credit Data” [25] are used again. They comprise a total of 20 features, encompassing 7 numerical and 13 categorical attributes along with an imbalanced binary target variable distinguishing nondefaulted cases (700 instances) from defaulted cases (300 instances). After applying one-hot encoding to the categorical features, we end up with a maximum of 48 binary features.

C. Formulation of clustering and randomly weighted MaxCut problem

In unsupervised learning, the problem of clustering data is defined as follows: Given a dataset $\mathbf{D}$ containing N data points, where each data point is represented as a feature vector x_i in a feature space $\mathbf{X}$: $D = \{x_1, x_2, \dots, x_N\}$, the goal of clustering is to partition the dataset into K clusters, where K is a user-specified or algorithmically determined parameter. Each cluster C_k is a subset of the dataset $\mathbf{D}$ with the following conditions:

- (1) Completeness: Every data point x_i belongs to exactly one cluster C_k , and the union of all clusters covers the entire dataset.
- (2) Homogeneity: Data points within the same cluster are similar to each other, and data points in different clusters are dissimilar. This is typically measured using a distance or similarity metric.
- (3) Minimization objective: The clusters are formed in a way that minimizes an objective function. This objective function may vary depending on the specific clustering algorithm being used.

Mathematically, the clustering problem can thus be defined as finding a partition of the dataset into K clusters that optimizes an objective function J , where

$$J = f(D, C_1, C_2, \dots, C_K). \quad (6)$$

The objective function J can be chosen based on the specific clustering algorithm’s criteria, such as minimizing intracluster distances or maximizing intercluster distances. The goal of clustering is to discover meaningful patterns or groupings within the data without any prior knowledge of the class labels. Various clustering algorithms, such as K-means [26], hierarchical clustering [27], and DBSCAN [28], can be used to solve this problem of finding the optimal partition of the dataset into clusters based on different criteria and distance measures.

The Euclidean distances for all data pairs x_i, x_j , i.e.,

$$w_{ij} = \|x_i - x_j\|_2, \quad (7)$$

can be interpreted as the elements of a symmetric matrix W , where all diagonal elements are zero. This matrix W can now be interpreted as an adjacency matrix of a graph G , where each vertex represents an element of W , and w_{ij} is the weight of the edge between the vertices i and j . In general, the graph is fully connected.

The common assumption in clustering—distant points belonging to different clusters—translates in the case of $K = 2$ clusters to the problem of maximizing the overall sum of all weights (distances) between nodes with different labels. Therefore, we come to a maximum-cut (MaxCut) problem—MaxCut(G, W) for the dense graph G of the distance matrix W [29,30]. A cut is a set of edges that separates all vertices V into two disjoint sets S and $\bar{S}$ with $S \cup \bar{S} = V$. The cost of a cut is given by the negative sum of all weights of edges connecting vertices in S with vertices in $\bar{S}$. With the introduction of binary variables $z_i \in \{0, 1\}$, $i = 1, \dots, N$, which denote whether x_i belongs to S or $\bar{S}$, the sum of all cuts can be expressed as the cost function

$$F_{\text{clustering}}(\mathbf{z}) = - \sum_{i,j=1}^N w_{ij} (z_i + z_j - 2z_i z_j). \quad (8)$$

Again, we end up with a QUBO problem. Here, we decided to use the “Moons” and “Blobs” datasets of SCIKIT-LEARN [31] with two features, i.e., $x \in \mathbb{R}^2$. Each data point then induces one binary decision variable. Using Moons and Blobs has the advantage that the clustering results can be either checked by, i.e., the K-means algorithm or visually.

In addition to the MaxCut problem that arises from the clustering problem formulation, we also consider MaxCut problems where the weights w_{ij} of the edges are randomly drawn from a uniform discrete distribution between 0 and 1000 in steps of 1.

In Appendix C, we show the distribution of nondiagonal and diagonal QUBO-matrix elements of the different optimization problems.

III. CLASSICAL OPTIMIZATION

To solve the optimization problems introduced above, we use the following methods or software solutions:

- (1) CPLEX [32];
- (2) GUROBI [33];
- (3) MQLib/Burer2002 Heuristic [34];
- (4) Goemans-Williamson’s algorithm [35].

Apart from MQLib/Burer2002, these classical optimization methods are all available in IBM’s QISKIT [36] for solving

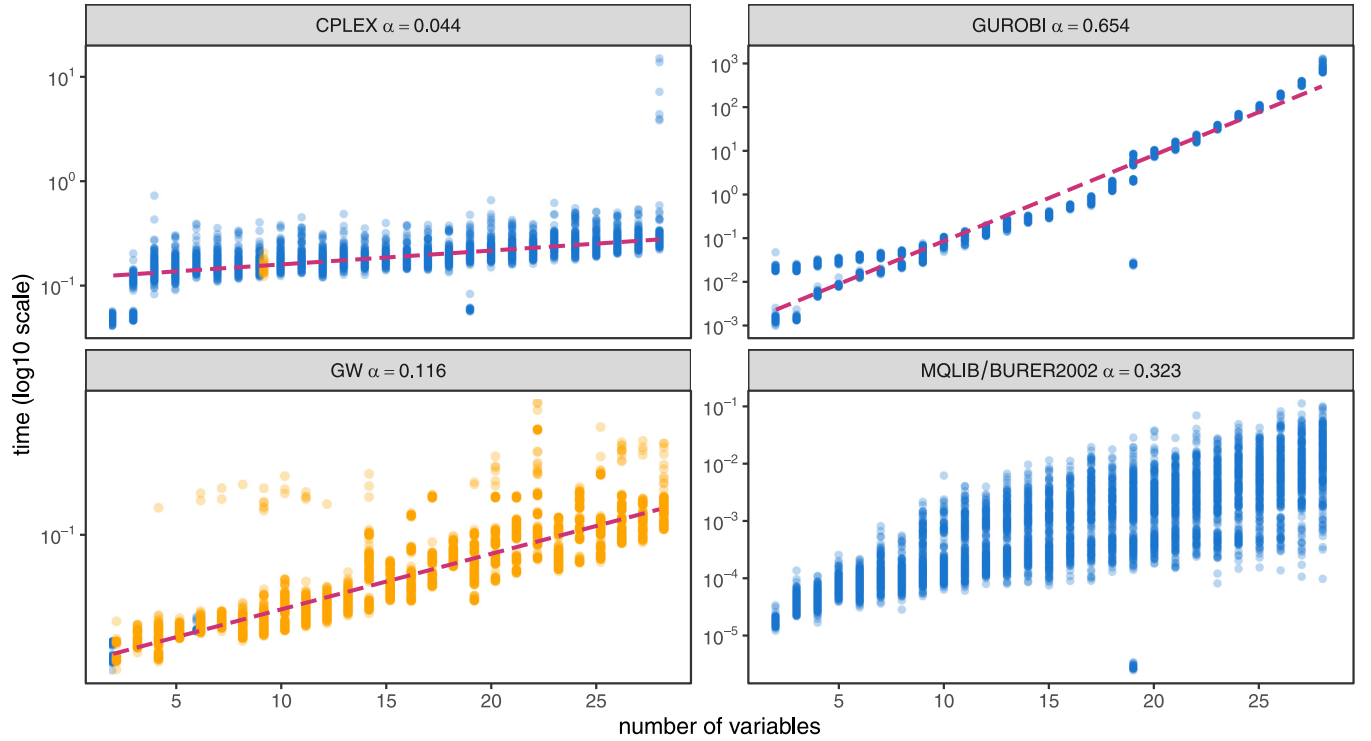


FIG. 1. Portfolio optimization: Runtime T (seconds) obtained with CPLEX, GUROBI, Goemans-Williamson, and MQLib/Burer2002 as a function of the number N of variables (bits). The dots represent the results for the ten problem instances, with blue indicating it was optimally solved, while orange indicates a nonoptimal solution. The slope α of a robust log2-linear model, i.e., $T \propto 2^{\alpha N}$ (red dashed line), is denoted in the header of each panel.

QUBOs with a uniform function call. Due to the large number of all possible solutions (2^N), a brute-force approach is infeasible for large N . Note that Goemans-Williamson's algorithm is an approximative algorithm; it does not guarantee to find the global optimum. In the literature, also more physics-inspired optimization methods are described, such as the parallel-tempering with isoenergetic cluster moves approach, see Ref. [37]. We refrained from implementing these approaches because our aim was primarily to use industry-established and ready-to-use optimization frameworks for our work.

Strictly speaking, none of the algorithms guarantees to find the global optimum, although at least CPLEX and GUROBI may provide additional information about the optimality of solutions. That is, they compute upper and lower bounds for the objective function value, which may indicate global optimality if the difference between both bounds is zero. MQLib/Burer2002, finally, does not provide information about the optimality of the solution. It continues trying to find better solutions until a user-defined maximum runtime is reached. At the end, it returns these solutions together with the times at which they have been found.

To investigate the scaling behavior of the different classical optimization methods, we perform the following experiments:

(1) Sample I instances with size N randomly from the respective dataset or distribution for each combinatorial optimization problem class. This accounts for any variation of the runtime caused by instance characteristics.

(2) For each instance, we measure R times how long each algorithm takes to find a solution. The repetition is intended to account for random variations of the runtime.

(3) Repeat this for each $N \in \{N_{\min}, N_{\min} + 1, \dots, N_{\max}\}$. This accounts for the influence of the problem size.

The specific settings considered for each problem class are

(1) Portfolio optimization: $I = 10$, $R = 20$, $N \in \{2, 3, \dots, 28\}$;

(2) Feature selection: $I = 20$, $R = 10$, $N \in \{2, 3, \dots, 48\}$;

(3) Clustering (Moons): $I = 20$, $R = 10$, $N \in \{2, 3, \dots, 100\}$;

(4) Clustering (Blobs): $I = 20$, $R = 10$, $N \in \{2, 3, \dots, 100\}$;

(5) MaxCut: $I = 10$, $R = 20$, $N \in \{11, 12, \dots, 29\}$.

In the following, we show the runtime measured during execution as a function of the number N of variables and the slope of the fit through the data. The slope is an estimator of the scaling factor, which depends on the dataset and on the algorithm.

In Fig. 1, the results for the portfolio optimization problem are shown. The absolute values for the runtime differ strongly between the four classical optimization schemes. On the one hand, GUROBI shows the longest absolute runtimes as well as the highest scaling factor. On the other hand, CPLEX shows the lowest scaling factor and MQLib/Burer2002 the shortest absolute runtime.

A similar behavior can be observed for the feature-selection problem, see Fig. 2. Again, CPLEX has the lowest scaling factor, MQLib/Burer2002 has the shortest absolute

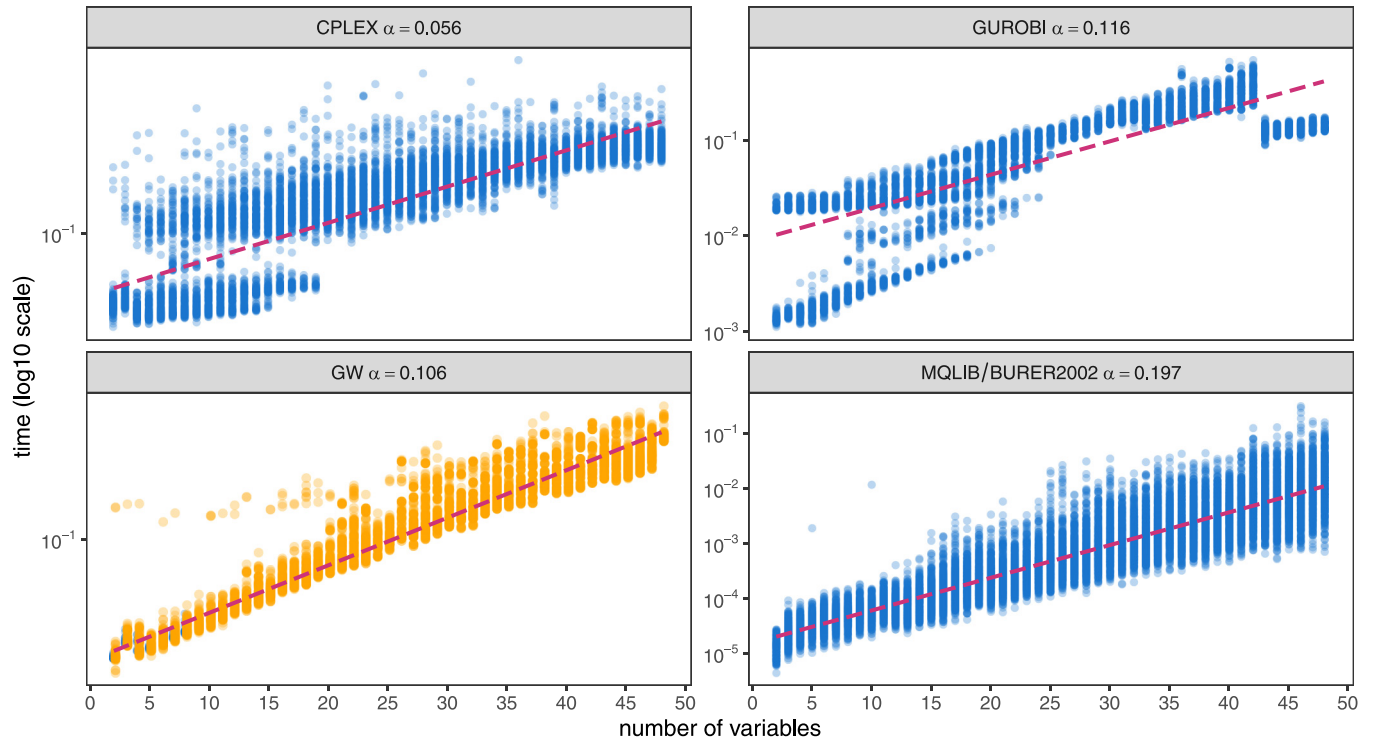


FIG. 2. Feature selection: Runtime T (seconds) obtained with CPLEX, GUROBI, Goemans-Williamson, and MQLib/Burer2002 as a function of the number N of variables (bits). The dots represent the results for the ten problem instances, with blue indicating it was optimally solved, while orange indicates a nonoptimal solution. The slope α of a robust log2-linear model, i.e., $T \propto 2^{\alpha N}$ (red dashed line), is denoted in the header of each panel.

runtimes, and Goemans-Williamson's (GW) algorithm does not find the optimal solution in most cases. However, MQLib/Burer2002 now has the highest scaling factor, and GW and CPLEX have the longest absolute runtimes.

In Fig. 3, the clustering results for the Moons dataset are shown. Interestingly, GW mostly finds the optimal solutions for this problem class, and GUROBI now has the lowest scaling factor, albeit also showing a strong discontinuity of runtimes at about $N = 45$. Similarly, but less pronounced, CPLEX also displays jumps of the runtime, e.g., around $N = 90$. GW has the largest scaling factor in this case. MQLib/Burer2002 has a marginally lower scaling factor than CPLEX, and is again the algorithm with the lowest absolute runtimes.

The results for the Blobs dataset are depicted in Fig. 13, which can be found in Appendix A. In this case, CPLEX has the lowest α , closely followed by GUROBI and MQLib/Burer2002 (with the lowest absolute runtimes). Besides some instances with very small N , CPLEX does not show the discontinuities displayed on the Moons data, while GUROBI has the same jump at about $N = 45$, in addition to jumps at lower values of N .

Compared with the other problem classes, the MaxCut instances seem to exhibit the least number of discontinuities, as shown in Fig. 4. GW only finds the optimal solution in some cases. In terms of our scaling estimates α of the other three algorithms, CPLEX performs best, followed by MQLib/Burer2002, while GUROBI has the largest scaling factor. In terms of absolute values, MQLib/Burer2002 has again the lowest absolute runtimes.

For each problem instance, we also performed separate runs with CPLEX (focusing on optimality), to confirm whether

we find globally optimal reference solutions in each case. CPLEX provides an upper bound of the gap to the global optimum, which can become zero, in which case the corresponding reference solution that was found is provably a global optimum. Overall, this gives us only a small number of instances for which we have no proof that the best solution we found is globally optimal:

- (1) Portfolio optimization: 1 of 270 instances;
- (2) Feature selection: 18 of 940 instances;
- (3) Clustering (Moons): 2 of 1980 instances;
- (4) Clustering (Blobs): 4 of 1980 instances;
- (5) MaxCut: 0 of 190 instances.

Moreover, in all cases, the relative gap between the best-found solution and the lower bound of the objective function value is smaller than 10^{-6} . Note that, in the Figs. 1–3, blue dots mark a run as solved even if we have no proof that the solution is a global optimum. Orange dots imply that the algorithm found a solution that is worse than the reference we determined.

In summary, it can be concluded that the various classical optimization methods exhibit different scaling behaviors on the investigated combinatorial optimization problems. For the subsequent comparison with quantum algorithms, we utilize the MQLib/Burer2002 algorithm for the following reasons: The Goemans-Williamson's algorithm is unsuitable for the comparison because it frequently does not find the optimum. The commercial optimization algorithms CPLEX and GUROBI are problematic because their search approach may change unexpectedly depending on various criteria, as is also evinced by the jumps and discontinuities of their runtime

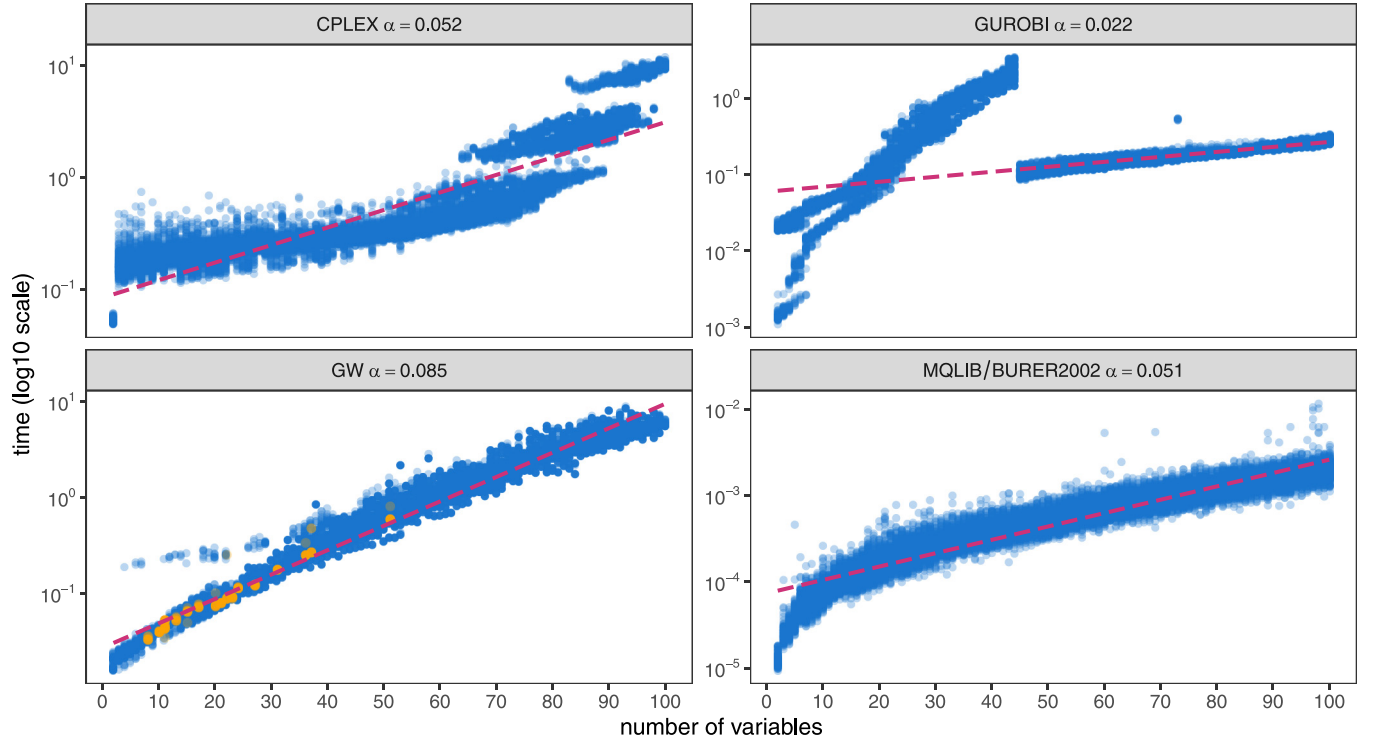


FIG. 3. Clustering (Moons): Runtime T (seconds) obtained with CPLEX, GUROBI, Goemans-Williamson, and MQLib/Burer2002, as a function of the number N of variables (bits). The dots represent the results for the ten problem instances, with blue indicating it was optimally solved, while orange indicates a nonoptimal solution. The slope α of a robust log2-linear model, i.e., $T \propto 2^{\alpha N}$ (red dashed line), is denoted in the header of each panel.

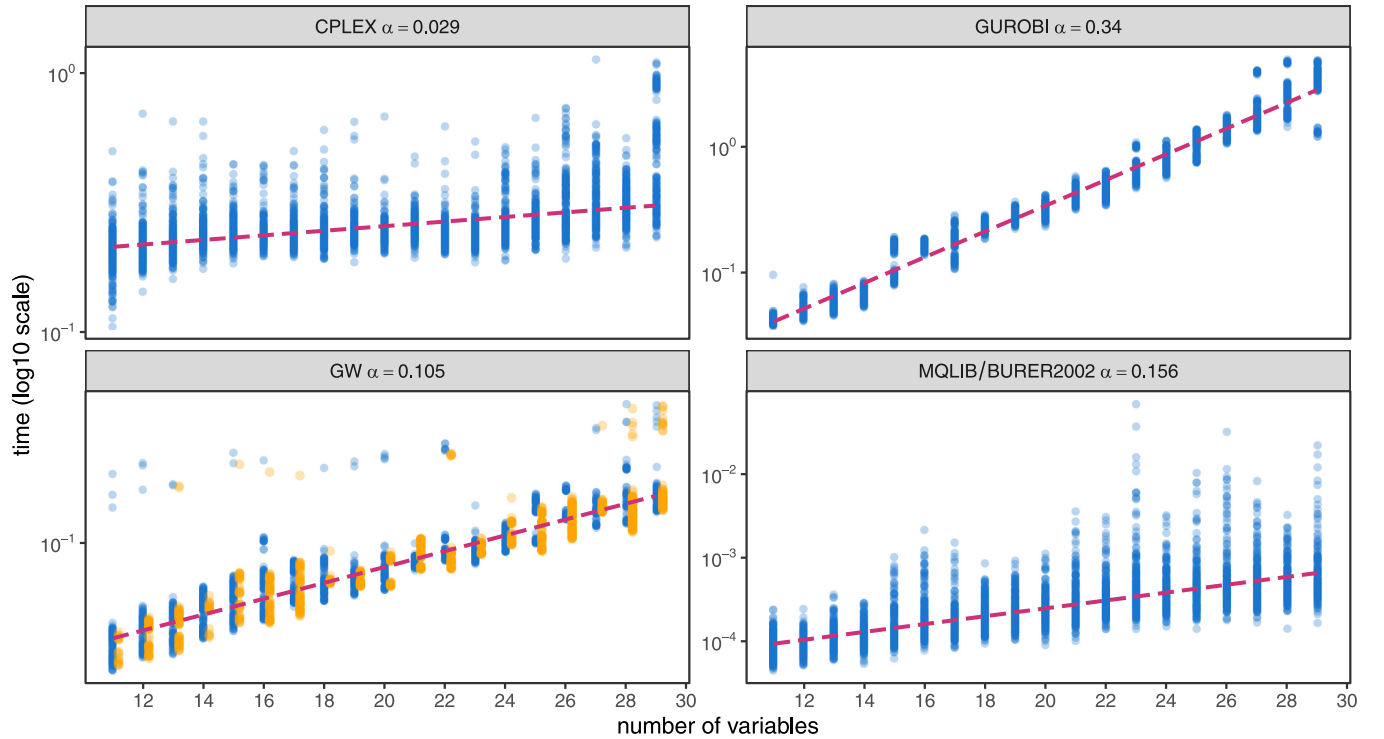


FIG. 4. MaxCut: Runtime T (seconds) obtained with CPLEX, GUROBI, Goemans-Williamson, and MQLib/Burer2002, as a function of the number N of variables (bits). The dots represent the results of individual algorithm runs on one instance, with blue indicating it was optimally solved, while orange indicates a nonoptimal solution. The slope α of a robust log2-linear model, i.e., $T \propto 2^{\alpha N}$ (red dashed line), is denoted in the header of each panel.

graphs, making it very hard to model their behavior. Moreover, the runtimes of CPLEX and GUROBI also include the efforts required for collecting additional information such as optimality bounds, which neither MQLib/Burer2002 nor the quantum algorithm discussed below provide. Especially for small problem sizes, the runtimes of CPLEX (of the order of 10^{-1} s) exhibit a considerable overhead compared with MQLib/Burer2002 (about 10^{-5} or 10^{-4} s). For these reasons, we argue that only the MQLib/Burer2002 algorithm reliably reflects the actual algorithmic complexity for finding the optimal solution and therefore appears to be the most suitable for comparison with the quantum algorithm.

IV. QUANTUM OPTIMIZATION

A. Linear-ramp QAOA

When using QAOA to solve the above-introduced optimization problems on a quantum computer, the following parametrized quantum state is generated:

$$|\psi_{\vec{\gamma}, \vec{\beta}}\rangle = \hat{U}_M(\beta_p) \hat{U}_F(\gamma_p) \cdots \hat{U}_M(\beta_1) \hat{U}_F(\gamma_1) |\psi_0\rangle_M, \quad (9)$$

with circuit parameters $\vec{\gamma} = (\gamma_1, \dots, \gamma_p)$ and $\vec{\beta} = (\beta_1, \dots, \beta_p)$, mixer unitary $\hat{U}_M(\beta_j) = e^{-i\beta_j \hat{M}}$ with transverse-field mixer $\hat{M} = -\sum_{i=1}^n \hat{X}_i$, and cost unitary $\hat{U}_F(\gamma_j) = e^{-i\gamma_j \hat{F}}$ with cost Hamiltonian $\hat{F}$ (defined by $\hat{F}|\mathbf{z}\rangle = F(\mathbf{z})|\mathbf{z}\rangle$ for basis states $|\mathbf{z}\rangle$). The initial state $|\psi_0\rangle_M$ corresponds to the mixer Hamiltonian $\hat{M}$, or more precisely, it is its eigenstate associated with its smallest eigenvalue. Hence, the state is the equal superposition state $|\psi_0\rangle = |+\rangle^{\otimes N}$ for this choice of $\hat{M}$. Finally, the probability of measuring the optimal solution $\mathbf{z}_{\text{opt}}$ is obtained as

$$P_{\text{opt}} = |\langle \mathbf{z}_{\text{opt}} | \psi_{\vec{\gamma}, \vec{\beta}} \rangle|^2. \quad (10)$$

In contrast with standard QAOA, where suitable values for the circuit parameters are suggested through a classical optimization routine, we sample them directly from the ansatz

$$\gamma_j = \Delta_\gamma x_j, \quad (11)$$

$$\beta_j = \Delta_\beta (1 - x_j), \quad (12)$$

with

$$x_j = \frac{j - \frac{1}{2}}{p}, \quad j = 1, 2, \dots, p, \quad (13)$$

where the parameters Δ_γ , Δ_β are left to be optimized. Additionally, we find optimized values for p , with the procedure outlined in more detail in Sec. IV B 4. Those three parameters characterize the linear-ramp schedule, which is illustrated in Fig. 5.

Intuitively, the above linear-ramp schedule can be motivated by the quantum adiabatic theorem [38], where the known ground state of $\hat{M}$ is transformed to the desired ground state of $\hat{F}$. Indeed, the above QAOA circuit, Eq. (9), reproduces the continuous adiabatic evolution in the limit $\Delta_\gamma, \Delta_\beta \rightarrow 0$ and $\Delta_\gamma p, \Delta_\beta p \rightarrow \infty$. As it turns out, however, the continuous limit is not necessary in order to prove convergence to the ground state (i.e., $P_{\text{opt}} \rightarrow 1$) in the limit $p \rightarrow \infty$: according to the discrete adiabatic theorem [39], this convergence is also guaranteed for finite $\Delta_\gamma, \Delta_\beta > 0$ —provided

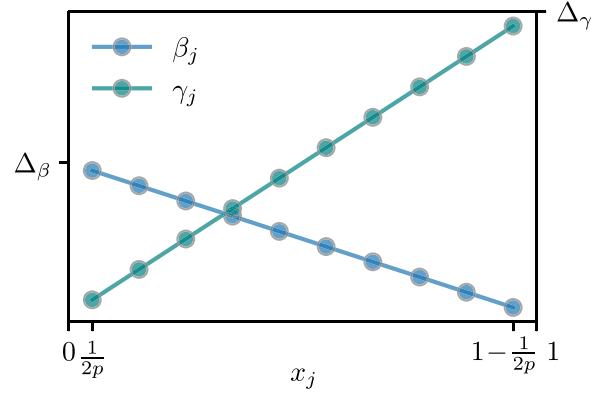


FIG. 5. Linear-ramp protocol according to Eqs. (11)–(13) (illustrated here for $p = 10$).

that Δ_γ , Δ_β do not exceed a critical value Δ_{crit} in order to prevent the eigenvalues of the unitary QAOA operators to “wrap around” the unit circle in the complex plane [13]. (In our case, $\Delta_{\gamma, \text{crit}} = 2\pi / (F_{\text{max}} - F_{\text{min}})$ and $\Delta_{\beta, \text{crit}} = \pi / N$, with F_{min} and F_{max} the smallest and largest eigenvalues of $\hat{F}$.)

From a practical point of view, however, we are rather interested in the behavior of LR-QAOA at finite p instead of $p \rightarrow \infty$. Even if the limit $P_{\text{opt}} \rightarrow 1$ is not reached in this case, the optimal solution can still be found by repeating the circuit several (approximately $1/P_{\text{opt}}$) times. In the following, we quantify the time required to find the optimal solution on a quantum computer in terms of the total depth D , defined as the depth of a single QAOA circuit (which, in turn, is proportional to p) times the number of circuit repetitions, see Eqs. (16) and (17) below.

B. LR extrapolation

We follow the idea to discard the variational optimization of the circuit parameters and find suitable values for Δ_γ , Δ_β , and p by extrapolating from smaller to larger problem sizes, corresponding to the number of qubits N . This enables us to make a prediction of the circuit parameters for larger problem sizes, while only the optimization over small problem sizes has to be performed. The method consists of three parts: the first part includes a reduction of the original problem to smaller subproblems, the second part describes the extrapolation, and the third part focuses on the postprocessing of the results.

1. Random sampling and grid search

For each QUBO instance of size $N \in \{12, \dots, 28\}$, we first create random subinstances of size $\tilde{N} \in \{4, 6, 8, 10\}$. This is done by randomly selecting $\tilde{N}$ out of the N binary variables and then constructing the QUBO cost functions as described in Sec. II. For feature selection, clustering and MaxCut, this simply amounts to choosing the corresponding submatrices of Q_{ij} , see Eq. (2). For portfolio optimization, the dependence of the budget constraint B on the problem size is taken into account (whereas the penalty term A is kept constant), see Eq. (4). Thereby, 10 random subinstances are created for each $\tilde{N}$.

In the next step, we use the 10 sampled subinstances of the smallest size ($\tilde{N} = 4$) and perform a grid search with an initial depth value $p_0 = 6$. That is, we construct a grid of $\log_{10}(\Delta_\gamma)$, $\log_{10}(\Delta_\beta)$ points, compute the probability of finding the optimal solution P_{opt} for each instance, and determine the average probability $\overline{P_{\text{opt}}}$ (averaged over the 10 subinstances). For all subproblem sizes, we choose an 11×11 grid with $\log_{10}(\Delta_\gamma)_{\min} \leq \log_{10}(\Delta_\gamma) \leq \log_{10}(\Delta_\gamma)_{\max}$ and $\log_{10}(\Delta_\beta)_{\min} \leq \log_{10}(\Delta_\beta) \leq \log_{10}(\Delta_\beta)_{\max}$. The initial size of the grid (for $\tilde{N} = 4$) is chosen as $\log_{10}(\Delta_\gamma)_{\max} - \log_{10}(\Delta_\gamma)_{\min} = \log_{10}(\Delta_\beta)_{\max} - \log_{10}(\Delta_\beta)_{\min} = 2$, centered around $[\log_{10}(\Delta_\beta)_{\min} + \log_{10}(\Delta_\beta)_{\max}]/2 = 0.5$ in case of $\log_{10}(\Delta_\beta)$, whereas the center of the $\log_{10}(\Delta_\gamma)$ -grid depends on the particular class of problem instance (due to different scalings of the corresponding QUBO matrices, see Appendix C). In particular, we choose for $[\log_{10}(\Delta_\gamma)_{\min} + \log_{10}(\Delta_\gamma)_{\max}]/2$ the values 0 for portfolio optimization and clustering, +2 for feature selection, and -2.5 for MaxCut. The logarithmic scale, i.e., considering $\log_{10}(\Delta_{\gamma,\beta})$ instead of $\Delta_{\gamma,\beta}$, is used to cover a sufficiently wide range of values.

2. Multivariate skew Gaussian fit and grid reduction

The above grid search provides a two-dimensional landscape consisting of $11 \times 11 = 121$ data points, see Fig. 6 as an example (for subproblem size $\tilde{N} = 10$ and depth $p = 107$). We can clearly see a dark blue region where the average probability $\overline{P_{\text{opt}}}$ assumes the largest values. It is the center of this region that we are interested in, and which we use for our extrapolation procedure below.

To determine the position of the center, we fit the discrete 11×11 -landscape by a continuous function. Empirically, we found that the following multivariate skew Gaussian function is well suited in order to approximate the $\overline{P_{\text{opt}}}$ -landscape (the detailed structure of which, in principle, depends on the random choice of subinstances) by a simple and smooth function

$$f_{\mu, \Sigma, \alpha, A, B}(\mathbf{x}) = A \exp\left(-\frac{1}{2}(\mathbf{x} - \mu)^T \Sigma^{-1}(\mathbf{x} - \mu)\right) \times \frac{1}{2} \left[1 + \operatorname{erf}\left(\alpha \cdot \frac{\mathbf{x} - \mu}{\sqrt{2}}\right) \right] + B, \quad (14)$$

where $\mathbf{x} = \{\log_{10}(\Delta_\gamma), \log_{10}(\Delta_\beta)\}$. The fit parameters consist of a two-dimensional vector μ (center of the Gaussian function), a 2×2 (inverse) covariance matrix $\Sigma^{-1} \geq 0$, a two-dimensional vector α characterizing the skewness, and two parameters $A, B \geq 0$. The maximum of this function is determined numerically and provides us with $\{\log_{10}(\Delta_{\gamma, \text{opt}}), \log_{10}(\Delta_{\beta, \text{opt}})\} = \mathbf{x}_{\text{opt}}$ and $\overline{P_{\text{opt}}}|_{\max} = f_{\mu, \Sigma, \alpha, A, B}(\mathbf{x}_{\text{opt}})$. Additionally, we extract the widths $s_\gamma = \sqrt{\Sigma_{1,1}}$ and $s_\beta = \sqrt{\Sigma_{2,2}}$ of the blue region from the diagonal elements of the covariance matrix Σ .

With these parameters, we set the new coordinates for the grid search of the next-sized subproblem in the following way:

$$\begin{aligned} \log_{10}(\Delta_\gamma)_{\min} &= \log_{10}(\Delta_{\gamma, \text{opt}}) - s_\gamma, \\ \log_{10}(\Delta_\gamma)_{\max} &= \log_{10}(\Delta_{\gamma, \text{opt}}) + s_\gamma, \\ \log_{10}(\Delta_\beta)_{\min} &= \log_{10}(\Delta_{\beta, \text{opt}}) - s_\beta, \\ \log_{10}(\Delta_\beta)_{\max} &= \log_{10}(\Delta_{\beta, \text{opt}}) + s_\beta, \end{aligned} \quad (15)$$

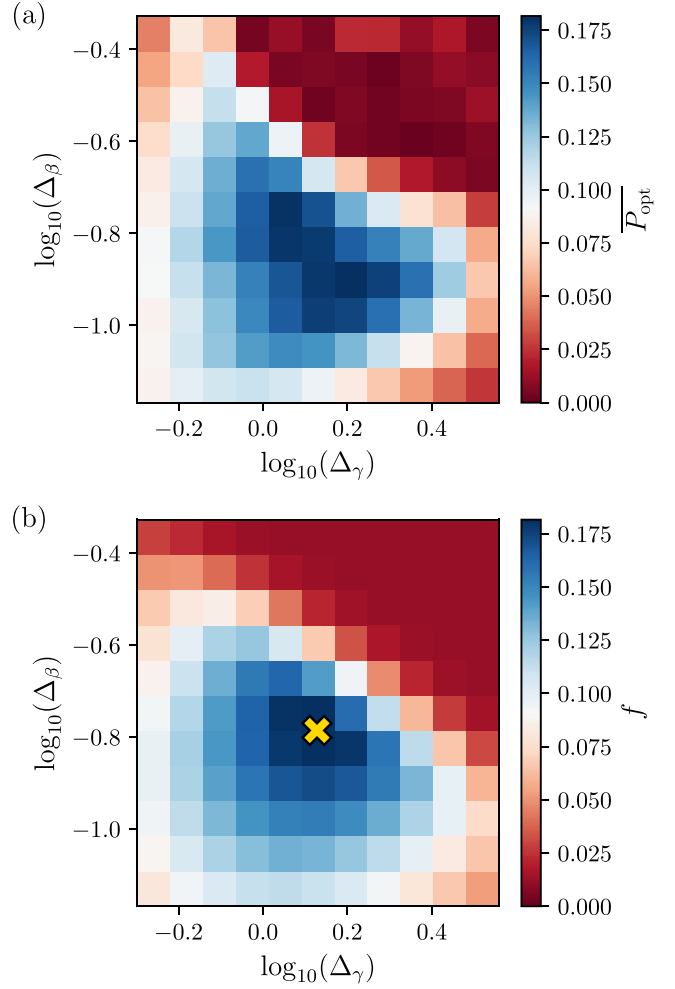


FIG. 6. (a) Portfolio optimization: Two-dimensional grid search landscape for 10 subproblems of size $\tilde{N} = 10$ sampled from a problem of size $N = 28$ at depth $p = 107$. The blue region shows the region with the highest average ground-state probability $\overline{P_{\text{opt}}}$. (b) Skewed Gaussian fit f , see Eq. (14), to the data shown in panel (a). The optimal values $[\log_{10}(\Delta_{\gamma, \text{opt}}), \log_{10}(\Delta_{\beta, \text{opt}})]$ are represented by the yellow cross.

which ensures that the optimum of the previous-sized problem is at the center of the following grid. This procedure is then repeated until we reach subproblem size $\tilde{N} = 10$.

3. Extrapolation

After having found optimized parameters $\{\log_{10}(\Delta_{\gamma, \text{opt}}), \log_{10}(\Delta_{\beta, \text{opt}})\}$ and $\overline{P_{\text{opt}}}|_{\max}$ for all sizes $\tilde{N} = 4, 6, 8, 10$ of the subinstances, we extrapolate these values toward the size N of the original instance. For this purpose, we assume a linear behavior on a log-log scale, i.e., we fit functions $y_i = a_i x + b_i$ to the above data points, where $x = \log_{10}(\tilde{N})$, $y_1 = \log_{10}(\Delta_{\gamma, \text{opt}})$, $y_2 = \log_{10}(\Delta_{\beta, \text{opt}})$, and $y_3 = \log_{10}(\overline{P_{\text{opt}}}|_{\max})$. Thereby, we obtain extrapolated values $\Delta_{\gamma, \text{extr}}$, $\Delta_{\beta, \text{extr}}$ of the linear-ramp circuit parameters together with an estimation $\overline{P_{\text{opt}, \text{extr}}}$ of the probability to measure the optimal solution of our problem instance with size N . In Fig. 7, this extrapolation procedure is illustrated for four different portfolio optimization instances of size $N = 28$, each at a certain depth

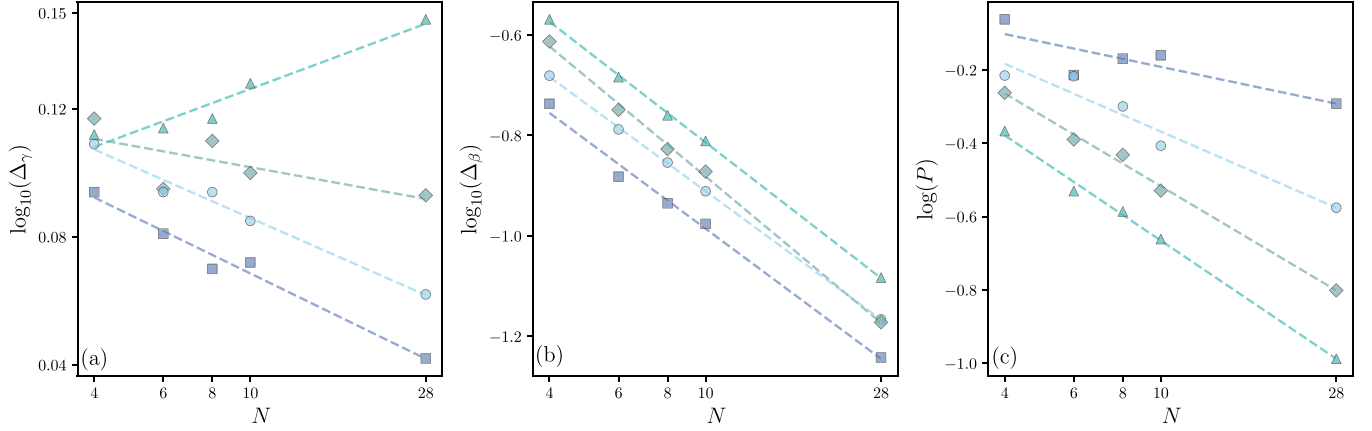


FIG. 7. Extrapolation of (a) $\log(\Delta_\gamma)$, (b) $\log(\Delta_\beta)$, and (c) $\log(P_{\text{opt}})$ for four instances of the portfolio optimization problem with $N = 28$ assets (triangles, diamonds, circles, squares). The symbols at $N = 4, 6, 8, 10$ indicate the optimized values found for the respective subproblems. The dashed lines represent linear fits (on log-log scale) through these points. Finally, the markers at $N = 28$ display the extrapolated values of these parameters. Each instance is processed at a certain depth: $p = 128$ (triangles), $p = 215$ (diamonds), $p = 304$ (circles), $p = 608$ (squares).

p (128, 215, 304, and 608). Each depth corresponds to the optimal depth for that instance, as determined in Sec. IV B 4 below.

At present, we do not have a theoretical justification for the algebraic scaling with respect to $\tilde{N}$ assumed above (i.e., linear behavior on a log-log scale), but we empirically found that it provides better results than exponential scaling, for example. To check the quality of our extrapolation, we have determined a 20×20 grid search landscape for one of the four instances shown in Fig. 7. Note that this grid search is numerically quite expensive, since it involves simulations of large quantum circuits (with $N = 28$ qubits)—in contrast with the extrapolation procedure based on simulations of small circuits (with $\tilde{N} = 4, 6, 8, 10$). As shown in Fig. 8, the estimated values $\Delta_{\gamma, \text{extr}}$, $\Delta_{\beta, \text{extr}}$ obtained from the extrapolation agree quite well with the true maximum. This example suggests that the predictions made using the extrapolation method provide a good estimate for the linear-ramp parameters. More data supporting this conclusion (at least for most instances) will be shown in Sec. V.

4. Total depth and optimization of p

As described above, the extrapolation procedure provides us with suitable estimates of the linear-ramp circuit parameters $\Delta_{\beta, \gamma}$ and the corresponding probability P_{opt} of measuring the optimal solution. What remains is to choose an appropriate value of the QAOA depth p . In principle, we may increase P_{opt} by increasing p , which, however, leads to longer circuits. To find an optimal value of p , we want to minimize the runtime of our quantum algorithm until the optimal solution is found. For this purpose, we concentrate on the depth d of the QAOA quantum circuit in the following. The corresponding time is then given by multiplying the depth by the time of a single quantum gate. Depending on the given quantum hardware, more specific estimates could be obtained, e.g., by taking into account different times of single-qubit and two-qubit gates or times of initial-state preparations and final-state measurements. We expect, however, that these details will not

significantly affect the scaling behavior in the limit of large problem sizes.

To determine the total quantum runtime, we therefore multiply the depth d of a single QAOA circuit (which, in turn is roughly proportional to the QAOA depth p) by the number N_{shots} of shots required to find the optimal solution:

$$D = dN_{\text{shots}}. \quad (16)$$

The median of the latter (defined by the condition that the probability of finding the optimal solution after N_{shots} circuit

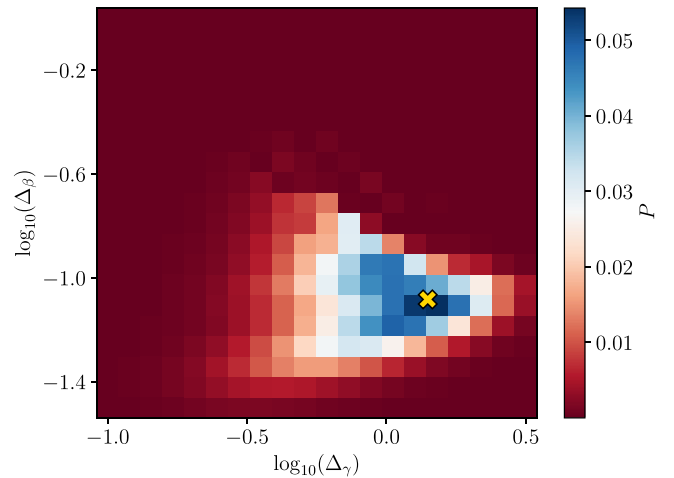


FIG. 8. Grid search landscape for one instance (corresponding to triangles in Fig. 7) with $N = 28$ qubits and QAOA depth $p = 128$. The optimal region is visualized by the dark-blue region. The optimal $(\Delta_{\gamma, \text{extr}}, \Delta_{\beta, \text{extr}})$ configuration predicted by the extrapolation method, represented by the yellow x symbol, is close to the optimal configuration found through grid search. Moreover, the corresponding maximum value of $P_{\text{opt}} \simeq 0.055$ (see the labels of the color bar on the right-hand side) is of the same order of magnitude as the extrapolated value of $P_{\text{opt, extr}} \simeq 0.103$. This indicates that the extrapolation method performs well.

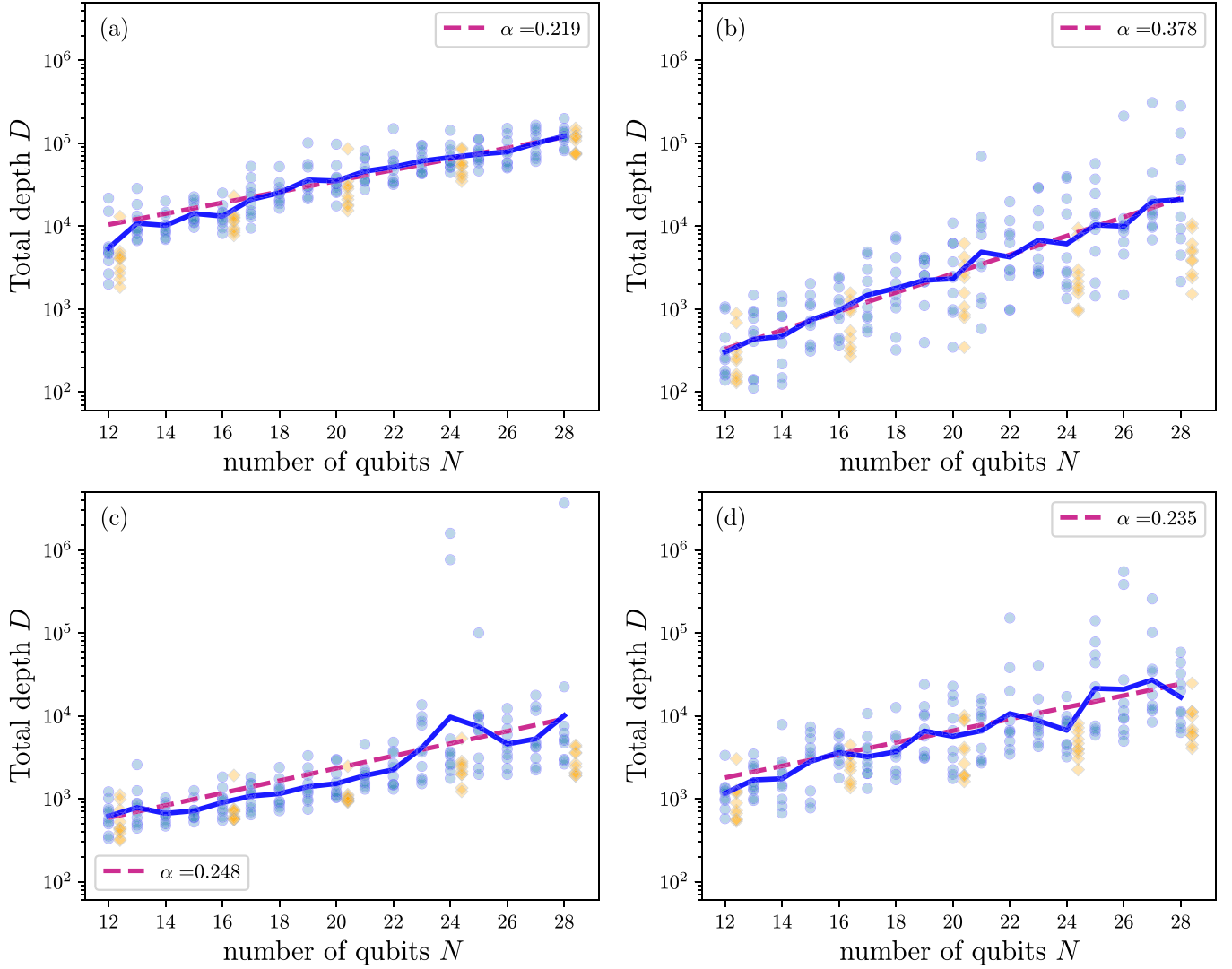


FIG. 9. Scaling of quantum runtime expressed as total depth D , see Eq. (16), with increasing problem size N (number of qubits) using linear-ramp QAOA with circuit parameters determined from extrapolation individually for each instance. The red dashed lines represent exponential fits with slope α (i.e., $D \propto 2^{\alpha N}$) of the geometric means (blue lines) over 10 problem instances per N (transparent blue dots). (a) Portfolio optimization. Compared with the classical runtime $\alpha_{\text{cl}} = 0.323$, see Fig. 1 (MQLib/Burer2002), the increase of the quantum runtime is slower ($\alpha = 0.219$), thus indicating a potential quantum scaling advantage. For the remaining three cases, (b) feature selection ($\alpha = 0.378$ vs $\alpha_{\text{cl}} = 0.197$), (c) clustering (Moons) ($\alpha = 0.248$ vs $\alpha_{\text{cl}} = 0.051$), and (d) MaxCut ($\alpha = 0.235$ vs $\alpha_{\text{cl}} = 0.156$), a scaling advantage compared with the classical runtime is not observed. The yellow diamond shapes represent the optimal values found by a minimizer, starting from the extrapolated parameters as an initial point. For better visibility, the diamonds have been slightly shifted along the x axis.

executions equals $1/2$) is given by

$$N_{\text{shots}} = \frac{\log(\frac{1}{2})}{\log(1 - P_{\text{opt}})}. \quad (17)$$

Alternatively, the expectation value given by $1/P_{\text{shots}}$ could also be used instead of the median.

The optimal QAOA depth p_{opt} can now be derived by minimizing D . More precisely, we minimize the estimated value of D based on the above extrapolation procedure, i.e., we set $P_{\text{opt}} = P_{\text{opt,extr}}$ in Eq. (17). We consider a logarithmically spaced grid of p -values $p \in \{1, 2, 3\} \cup \{2^{i/4}\}$, $i = 9, 10, 11, \dots$ and, starting from an initial value p_0 (for which we take the optimal value found for the previously calculated instance, or $p_0 = 6$ in case of the first instance corresponding

to the smallest problem size $N = 12$), we increase or decrease p on this grid until a local minimum of D is found.

V. RESULTS AND DISCUSSION

In this section, we analyze the LR-QAOA runtime scaling for the four use cases (portfolio optimization, feature selection, clustering, and MaxCut) introduced in Sec. II. We increase the problem size from $N = 12$ to $N = 28$. For each N , 10 randomly chosen problem instances are generated, and LR-QAOA circuit parameters (Δ_γ , Δ_β , and p) are found using the extrapolation procedure described in Sec. IV A. Finally, the total depth D —which is proportional to the quantum runtime for finding the optimal solution (see Sec. IV B 4)—is determined by Eqs. (9), (10), (16), and (17) (using a noiseless

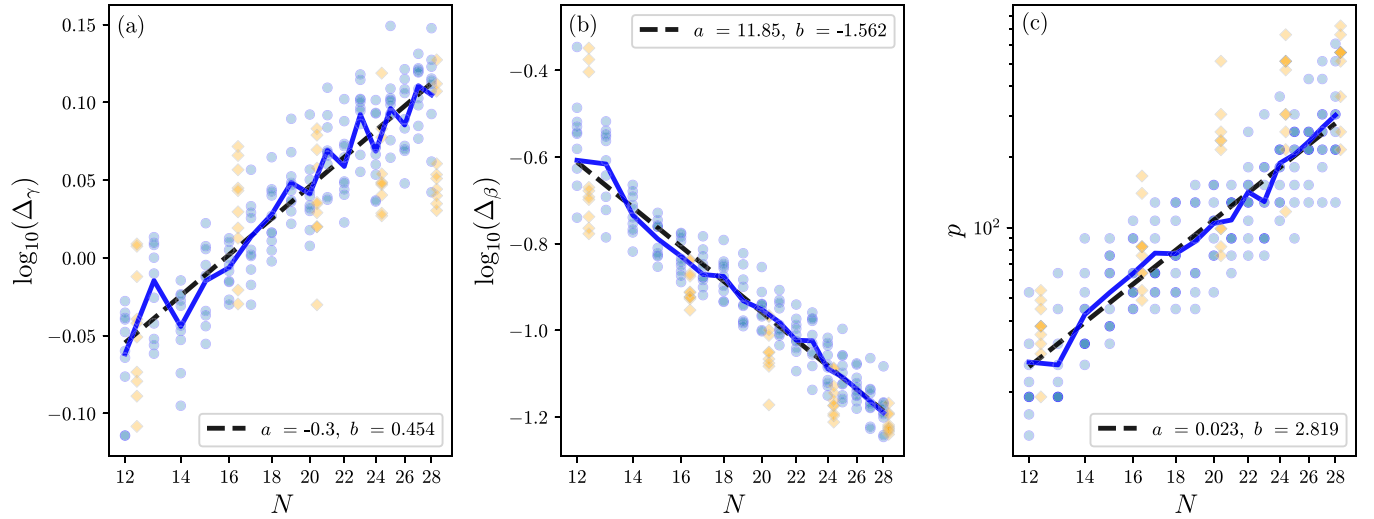


FIG. 10. Portfolio optimization: optimized linear-ramp parameters (a) Δ_γ , (b) Δ_β , and (c) QAOA depth p as a function of number of qubits N , depicted as a log-log plot, for the same instances as in Fig. 9(a). The transparent blue dots represent values obtained from extrapolation and the blue lines show the $1/D$ -weighted geometric means (see main text) over all instances. The values for Δ_γ as well as for p are increasing with increasing N while the values for Δ_β are decreasing. The yellow diamonds represent the optimal values found by a minimizer [as in Fig. 9(a)] and have been shifted along the x axis for better visibility. The black dashed lines display algebraic fits $f(N) = aN^b$ to the $1/D$ -weighted geometric means and determine the universal parameter scaling used in Fig. 11(a).

quantum emulator). Note that the last step is the only one that involves the execution of a quantum circuit with larger size ($N > 10$). Finally, we determine the runtime scaling with increasing N and compare it with the classical result (MQLib/Burer2002) of Sec. III.

A. LR-QAOA runtime scaling

For the case of portfolio optimization, the results for the individual 10 random instances per N are displayed by the blue circles in Fig. 9(a). To extract the scaling behavior, we perform an exponential fit, $D \propto 2^{\alpha N}$ (red dashed line). With this, we find a scaling exponent for the runtime of $\alpha \approx 0.219$, which is smaller than the classical scaling exponent of $\alpha \approx 0.323$ (see Fig. 1, bottom right). This indicates a potential quantum advantage regarding the runtime scaling.

To test our extrapolation procedure, we further optimized the circuit parameters (Δ_γ , Δ_β , and p)—starting from the extrapolated parameters as initial points—by directly minimizing the actual total depth D (instead of its estimate based on extrapolation). Since this involves many executions of large quantum circuits (up to $N = 28$) for each instance, we restricted ourselves to every fourth problem size $N = 12, 16, \dots$. All in all, we see that the depths D achieved with the extrapolated parameters (blue circles) are remarkably close to the actual minima (yellow diamonds) in Fig. 9(a). This underlines the effectiveness of the extrapolation scheme.

Figure 9(b) shows the results for the feature-selection problem, which we evaluate on the same footing as portfolio optimization. Here, we find a scaling for the quantum runtime of $\approx 2^{0.378N}$. In comparison, the classical runtime exhibits a scaling of $\approx 2^{0.197N}$ (see Fig. 2, bottom right), indicating that this combinatorial optimization problem is faster to solve classically.

A similar picture is observed for the clustering problem (Moons), as shown in Fig. 9(c), and the MaxCut problem, see Fig. 9(d). In these cases, we observe a quantum runtime scaling of $\approx 2^{0.248N}$ for clustering and $\approx 2^{0.235N}$ for MaxCut. In comparison, the classical heuristic MQLib/Burer2002 exhibits a scaling of $\approx 2^{0.051N}$ for clustering (see Fig. 3, bottom right) and $\approx 2^{0.156N}$ for MaxCut (see Fig. 4, bottom right), indicating that the quantum algorithm takes significantly longer to solve the problem. For clustering, we also observe some outliers with extremely large values of D (e.g., three instances with $D \approx 10^6$ at $N = 24$ and 28). This suggests that, especially for the clustering problem, the extrapolation scheme is less stable than, e.g., in the case of portfolio optimization and fails in providing good circuit parameters for some instances. This is also confirmed by the results of the direct optimization (yellow diamonds) in Fig. 9(c), which do not exhibit such extreme outliers.

B. Universal parameter scaling

To reduce these outliers, we additionally implemented an approach similar to the one proposed in Ref. [18]: rather than trying to find LR-QAOA circuit parameters individually for every single instance, the idea is to use the same “universal” parameters for different instances.

Figure 10 shows the circuit parameters obtained from our extrapolation procedure (blue circles) for the case of portfolio optimization. To determine the universal parameter scaling, we first evaluate—for each of the three parameters $y = \log_{10}(\Delta_\gamma)$, $\log_{10}(\Delta_\beta)$, or $\log_{10}(p)$, respectively—the $1/D$ -weighted average (blue lines), i.e., $\langle y \rangle = (\sum_i \frac{y_i}{D_i}) / (\sum_i \frac{1}{D_i})$, where the sum over i refers to the 10 instances per N , and the weight $1/D_i$ is employed to suppress those cases where the extrapolation fails in providing good circuit parameters (leading to a large total depth D_i for instance i). Remarkably, we

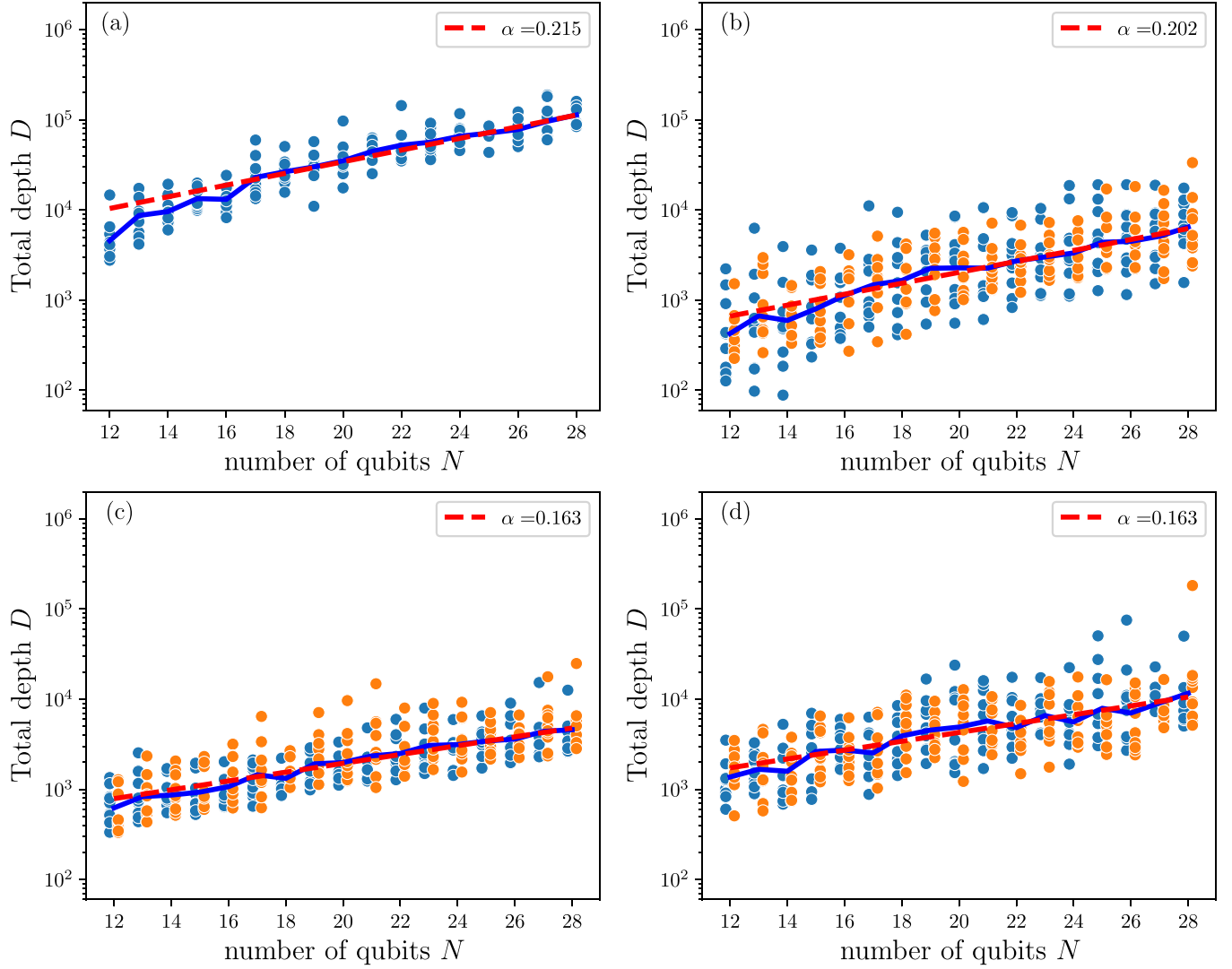


FIG. 11. Scaling of quantum runtime expressed as total depth D , see Eq. (16), with increasing problem size N (number of qubits) using linear-ramp QAOA with universal circuit parameters (defined by the algebraic fits in Figs. 10 and 14–16, black dashed lines). The blue circles refer to the same 10 problem instances per N as in Fig. 15, whereas the orange circles in panels (b)–(d) correspond to 10 new randomly chosen problem instances per N . As in Fig. 9, the red dashed lines represent exponential fits with slope α (i.e., $D \propto 2^{\alpha N}$) of the geometric means (blue lines) over all problem instances per N . (a) Portfolio optimization. The result is almost identical to the one in Fig. 9, exhibiting a scaling advantage compared with the classical benchmark ($\alpha = 0.22$ vs $\alpha_{\text{cl}} = 0.323$). For the remaining three problem classes (b) feature selection, (c) clustering (Moons), and (d) fully connected weighted random MaxCut, the choice of universal parameters significantly improves the LR-QAOA scaling coefficients as compared with Fig. 9. For (b) feature selection and (d) MaxCut, equality with the classical benchmark ($\alpha_{\text{cl}} = 0.197$ and 0.156 , respectively) is achieved. For clustering (Moons), the classical benchmark ($\alpha_{\text{cl}} = 0.051$) is not reached.

see that these averages closely follow algebraic fits using the function $f(N) = aN^b$ (with $f = \Delta_\gamma$, Δ_β , or p , respectively); see the dashed lines in Fig. 10.

These algebraic fits are now taken to define our universal parameter scaling. In other words, we again simulate all instances shown in Fig. 9(a)—but now using LR-QAOA circuit parameters given by the black dashed line instead of the blue points displayed in Fig. 10(a). For portfolio optimization, the result shown in Fig. 11(a) is similar to the previous one in Fig. 9(a). In particular, the rate of the exponential increase $\simeq 2^{0.215N}$ is almost unchanged.

For the other three problem classes, however, the results in Figs. 11(b)–11(d) reveal clear improvements when using the

universal parameter scaling (displayed by the black dashed lines in Figs. 14–16, see Appendix B). In contrast with Figs. 9(b)–9(d), strong outliers exhibiting extremely large depths D are no longer present. Moreover, we generated, for each problem size N , ten additional random instances (orange circles) and evaluated them using the same universal parameter scaling as for the original ten instances (blue circles), with similar results. The scaling coefficients α (determined from all 20 instances per N) are significantly smaller than in Fig. 9 before. For feature selection and the fully connected weighted random MaxCut problem, see Figs. 11(b) and 11(d), almost equality with the classical benchmarks ($\alpha_{\text{cl}} = 0.197$ and 0.156 , respectively) is now achieved, whereas, for clustering

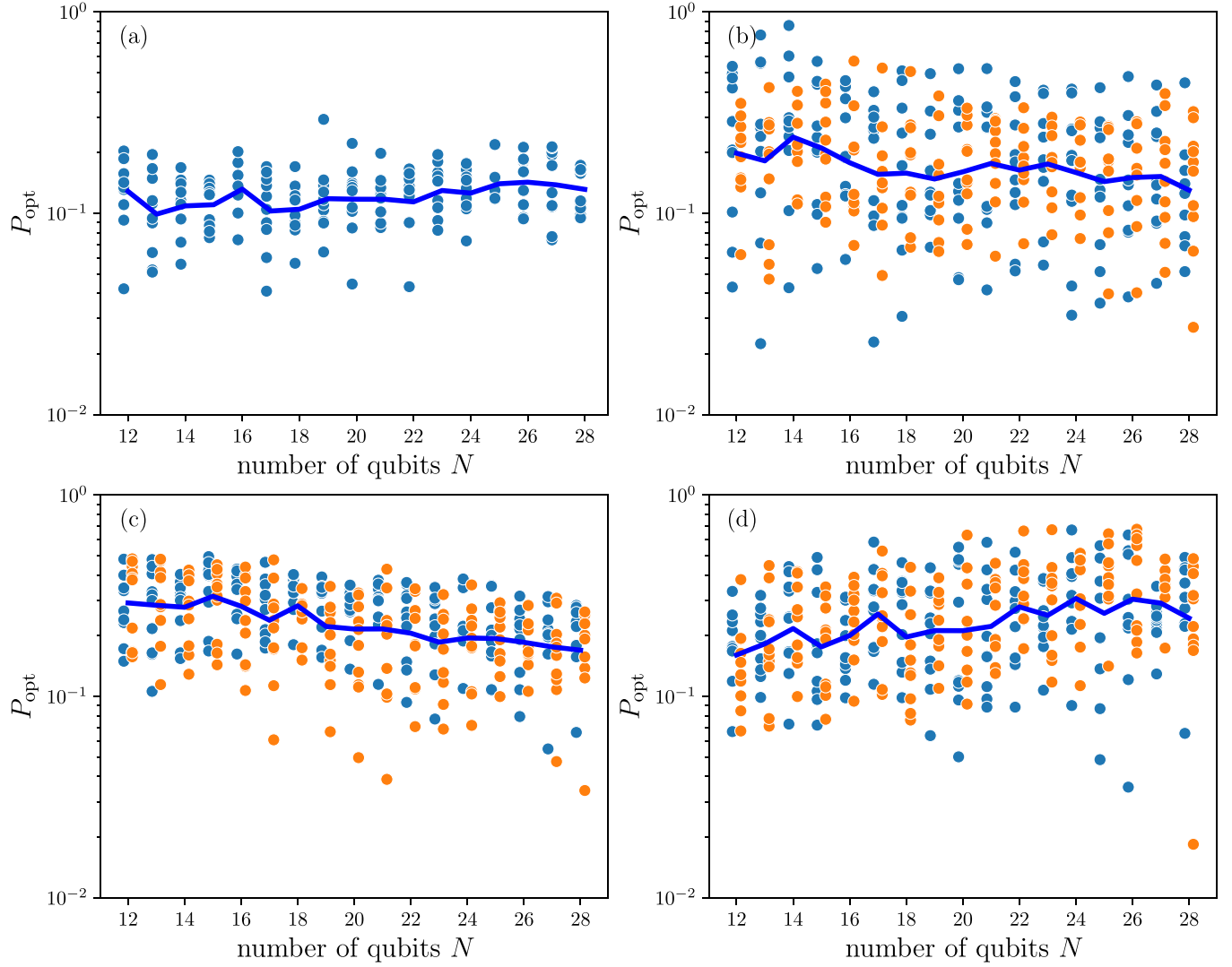


FIG. 12. Probability P_{opt} to measure the optimal solution after a single run of the LR-QAOA circuit for the same instances and the same universal parameters as in Fig. 11 (blue and orange circles). The (geometric) averages $\langle P_{\text{opt}} \rangle$ over all problem instances are displayed by the blue lines. On average, the probabilities $\langle P_{\text{opt}} \rangle$ remain above 0.1 and appear to be constant as a function of N in the case of (a) portfolio optimization and (d) fully connected randomly weighted MaxCut, whereas, for (b) feature selection and (c) clustering, a slight decrease of $\langle P_{\text{opt}} \rangle$ with increasing N is observed.

(Moons), the classical scaling ($\alpha_{\text{cl}} = 0.051$) is still better. All in all, we observe that, for our four problem classes, the scaling coefficients using LR-QAOA with universal parameter scaling are remarkably similar to each other (ranging between $\alpha = 0.16$ and 0.22), whereas the classical benchmarks exhibit larger variations (between $\alpha_{\text{cl}} = 0.051$ and 0.323).

Concerning the weighted MaxCut problem, a more advantageous scaling coefficient ($\alpha = 0.107$) has been observed in Ref. [18]. As we point out in Appendix D, this difference can be traced back to the selection of classically “hard” problem instances performed in Ref. [18].

Finally, we display in Fig. 12 the probabilities P_{opt} of measuring the optimal solution for the same instances and same LR-QAOA parameter scaling as in Fig. 11. In all cases, the probabilities are, on average, above 0.1. For portfolio optimization and MaxCut [Figs. 12(a) and 12(d)], the probability appears (on average) to remain constant (or even slightly in-

crease in the case of MaxCut) as a function of the problem size N . For feature selection and clustering (Moons), we observe a slight decrease. We note that the behavior of P_{opt} with increasing N is relevant for the following reason: Since, according to our universal parameter scaling, the QAOA depth $p(N) = aN^b$ scales algebraically and the depth d of the QAOA circuit is proportional to pN , constant median P_{opt} would imply algebraic, i.e., subexponential, scaling of the median quantum runtime as quantified by the total depth D , see Eq. (16).

VI. CONCLUSION

In this work, we present an extrapolation method to optimize linear-ramp QAOA parameters, enabling us to discard the need for variational optimization employed in standard QAOA. We start with the introduction of four different combinatorial optimization use cases: portfolio optimization, feature

selection, clustering, and weighted MaxCut, which are subsequently solved using classical methods (CPLEX, GUROBI, MQLib Burer2002, Goemans-Williamson) to assess the classical runtime. The scaling of the runtime is investigated by fitting the runtime with an exponential model, which serves as a benchmark for the runtimes of our quantum algorithm. In particular, we concentrate on the runtime scaling of the MQLib Burer2002 heuristics, since it exhibits the lowest absolute values of the runtime together with a well-defined scaling behavior over a large range of problem sizes.

Our quantum algorithm is based on linear-ramp QAOA, for which the circuit parameters are determined by a linear ansatz. This ansatz is characterized by three parameters (Δ_γ , Δ_β , and depth p), which have to be optimized. In this paper, we expose a method to derive those optimized parameters through extrapolation from smaller to larger problem sizes. In particular, the extrapolation is based on optimizations of QAOA circuits with reduced size ($\tilde{N} = 4, 6, 8, 10$) corresponding to randomly sampled subinstances of the original problem instance (with size N up to 28). The quantum runtime is then expressed in terms of the total depth $D = dN_{\text{shots}}$, i.e., the depth d of a single linear-ramp QAOA circuit times the (median of) the number of shots required to measure the optimal solution on a noiseless quantum emulator. Comparing quantum and classical runtime scalings, we find an indication of quantum advantage for one of the three use cases mentioned above, i.e., for portfolio optimization.

We furthermore showed that, based on the extrapolated optimized circuit parameters, a simple ansatz can be derived, according to which these parameters can be scaled algebraically with increasing problem size N , independently of the particular problem instance at hand. Employing this universal parameter scaling leaves the advantageous results in case of portfolio optimization unchanged, but significantly improves the scaling of quantum runtime in case of the three other problem classes (achieving equality with the classical benchmark in case of feature selection and MaxCut).

One might argue that the cost of finding the optimized circuit parameters should also be taken into account in the evaluation of the quantum runtime scaling. Although, as mentioned above, our extrapolation scheme is based on the analysis of small subproblems with fixed size (up to $\tilde{N} = 10$, independently of N), the computational time required for this analysis nevertheless increases with increasing problem size N , due to the fact that the optimal values of the QAOA depth p tend to increase for larger N . However, the idea of the present paper is rather to analyze the general potential of LR-QAOA, assuming that suitable circuit parameters are already known or can be efficiently found. An important insight provided by our article is that methods based on extrapolation and scaling of circuit parameters with increasing problem size appear to be a promising tool for this purpose, and the results presented here may inspire further development of such methods.

Probably the most important open topic concerns the question of whether quantum optimization admits polynomial-time solutions of some instances of combinatorial optimization problems which, classically, can be solved only in exponential time. This possibility is suggested by our observation that, for portfolio optimization and MaxCut, the median

probability P_{opt} of finding the optimal solution after a single execution of the LR-QAOA circuit does not decrease with increasing N , while, at the same time, the depth of the circuit scales algebraically. If true also for larger N , this would imply polynomial scaling of the median quantum runtime, i.e., at least half of all problem instances in the respective ensemble could be solved efficiently.

To answer this question, a larger range of problem sizes (beyond $N = 28$ as in the present paper) must be addressed in future work, which will either require extensive quantum simulations using high-performance computers and advanced simulation techniques or real, fault-tolerant quantum hardware with a sufficient number of qubits (about 50 to 100 or more).

Furthermore, a more elaborate statistical analysis, not concentrating on the median runtime as we did, but also considering the scaling of different percentiles [40], would provide additional insight. Finally, a deeper understanding of the structures of combinatorial optimization problems, which determine the hardness of their solution by classical or quantum methods, constitutes another challenging goal for future research.

ACKNOWLEDGMENTS

This work is funded by the Ministry of Economic Affairs, Labour and Tourism Baden-Württemberg in the frame of the Competence Center Quantum Computing Baden-Württemberg (Projects No. QORA II and No. KQCBW24) and by the European Union's HORIZON Europe program via project SPINUS (No. 101135699). A special thanks goes to Michael Krebsbach for his valuable support.

DATA AVAILABILITY

The data that support the findings of this article are openly available [41].

APPENDIX A: ADDITIONAL CLASSICAL RESULTS

Figure 13 illustrates the runtime for the clustering problem using the Blobs dataset. Employing a different dataset results in slightly varying outcomes. This emphasizes the influence of both the dataset and the selected instances on runtime scaling. Notably, the Goemans-Williamson algorithm is able to find the optimal solution in this case, and exhibits a better scaling factor α (slope of exponential fit) compared with the Moons dataset.

APPENDIX B: LR-QAOA PARAMETERS FOR FEATURE SELECTION, CLUSTERING, AND MAXCUT

Figure 14 shows the same parameters for the feature selection problem as shown in the main text (Fig. 10) for portfolio optimization. In this case, Δ_γ and Δ_β remain relatively constant as the problem size increases.

The parameters for the clustering problem are illustrated in Fig. 15, again showing a different trend compared with the other problems. Here, we see Δ_γ decreasing and Δ_β increasing. Finally, the parameters for the MaxCut problem are shown in Fig. 16.

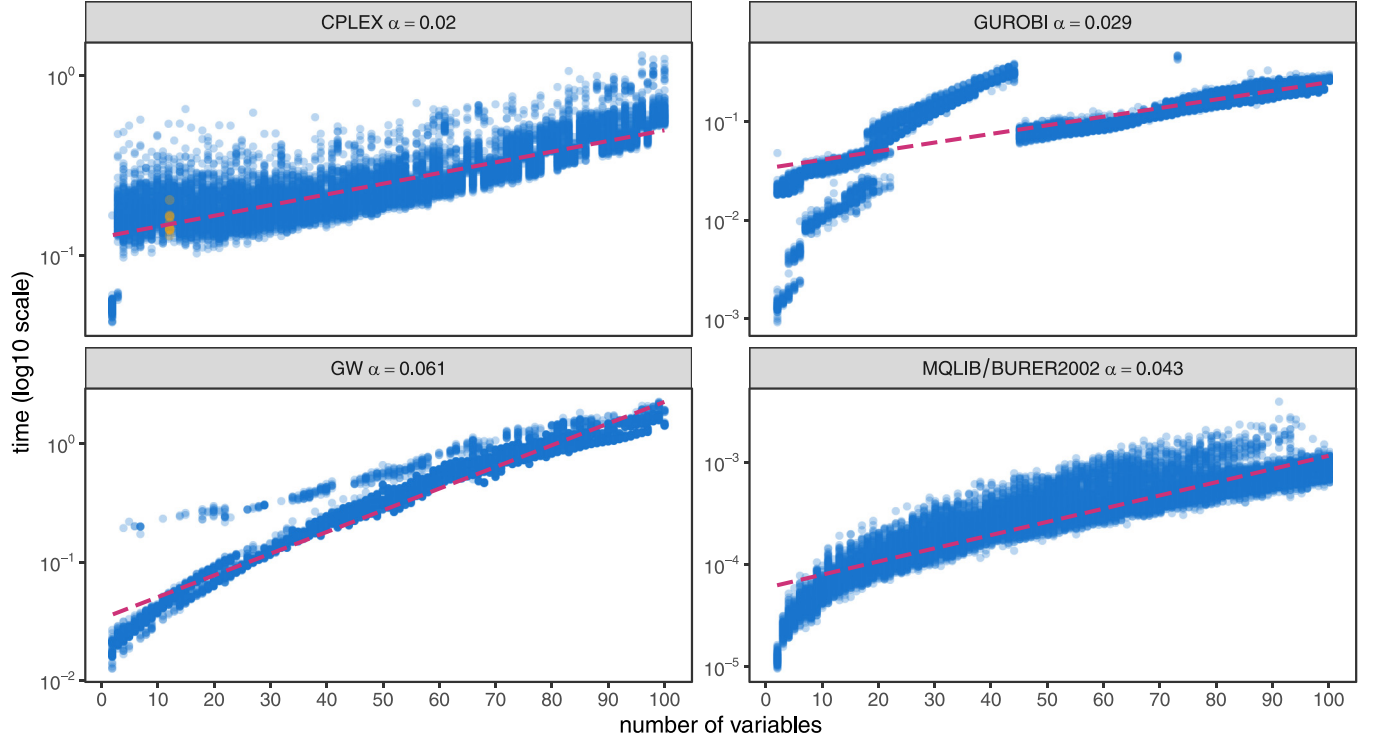


FIG. 13. Clustering (Blobs): Runtime T (seconds) obtained with CPLEX, GUROBI, Goemans-Williamson, and MQLib/Burer2002, as a function of the number n of variables (bits). The dots represent the results for the ten problem instances, with blue indicating it was optimally solved, while orange indicates a nonoptimal solution. The slope α of a robust $\log_2$ -linear model, i.e., $T \propto 2^{\alpha n}$ (red dashed line), is denoted in the header of each panel.

APPENDIX C: DISTRIBUTION OF THE QUBO DATA

All optimization problems considered above are defined by their corresponding QUBO matrices. To compare them, we performed the following analysis: For a given problem size (here, $N = 20$) and for each problem, we calculated the

maximum absolute value across all instances and normalized the respective QUBO matrix by this value. Subsequently, we computed the box plots for both the diagonal and nondiagonal elements. In Fig. 17, we compare the box plots of the portfolio optimization problem with those of the other problems: The distributions of both the diagonal and nondiagonal elements

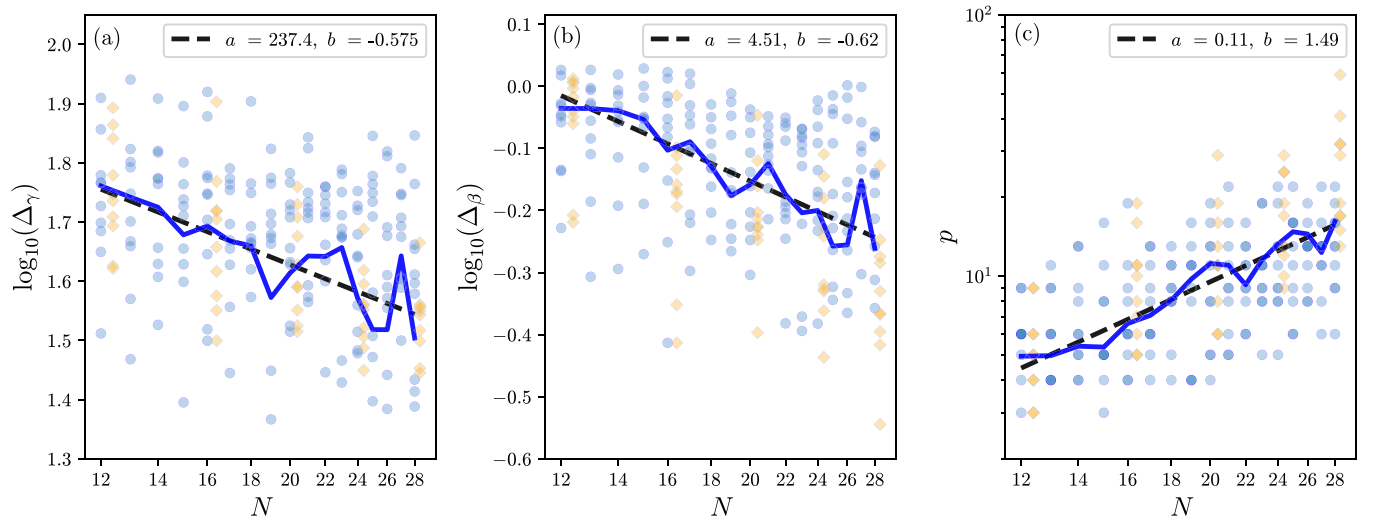


FIG. 14. Feature selection: same representation as for portfolio optimization of optimized linear-ramp parameters (a) Δ_γ , (b) Δ_β , and (c) QAOA depth p as a function of number of qubits N on a log-log scale. In this case, the logarithmic values for Δ_γ and for Δ_β are relatively constant with N . For the depth p , we observe a nearly linear scaling with respect to N . The yellow diamonds again represent the optima found by a minimizer. The black dashed lines display algebraic fits $f(N) = aN^b$ to the $1/D$ -weighted geometric means (blue lines) and determine the universal parameter scaling used in Fig. 11(b).

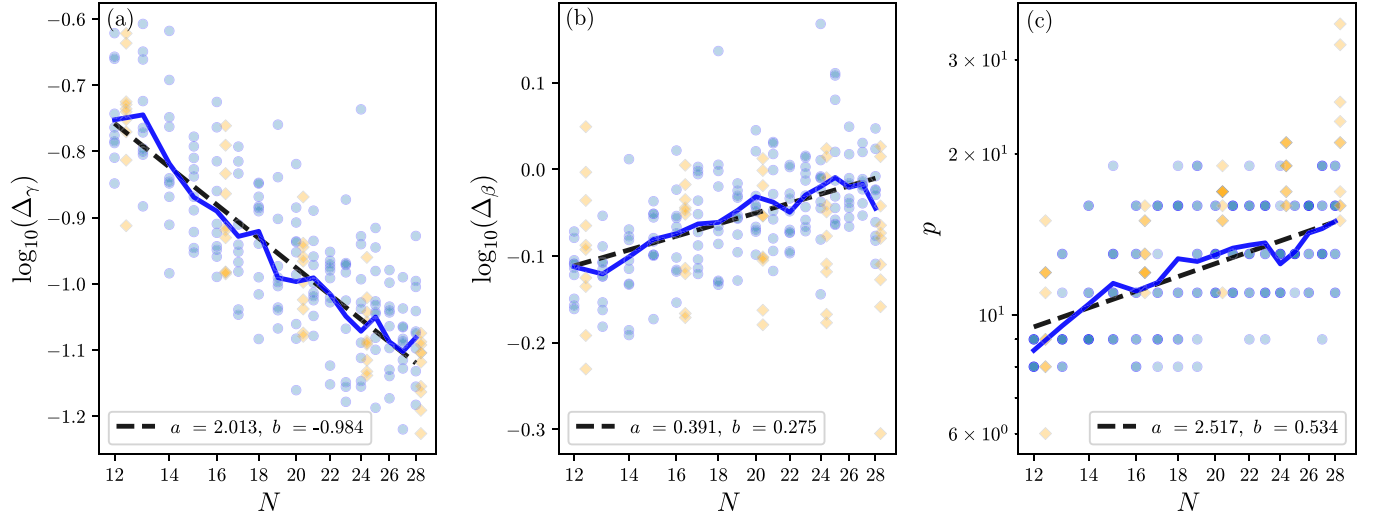


FIG. 15. Clustering (Moons): optimized linear-ramp parameters (a) Δ_γ , (b) Δ_β , and (c) QAOA depth p as a function of number of qubits N on a log-log scale. In this case, the values for Δ_γ are decreasing while the values for Δ_β are slightly increasing with N . For the depth p , we observe a scaling of $\propto \sqrt{N}$. The optima found by a minimizer are represented by the yellow diamonds. The black dashed lines display algebraic fits $f(N) = aN^b$ to the $1/D$ -weighted geometric means (blue lines) and determine the universal parameter scaling used in Fig. 11(c).

of the portfolio optimization QUBO are much narrower than those of the other problems. This is due to the fact that portfolio optimization includes a penalty term to enforce the constraint, i.e., the maximum number of assets allowed in the portfolio.

APPENDIX D: RESULTS FOR HARD MAXCUT INSTANCES

Concerning the weighted random MaxCut problem, our result ($\alpha = 0.163$) for the scaling coefficient, see Fig. 11(d), differs from the one ($\alpha = 0.107$) obtained in Ref. [18]. The latter result has been obtained for a selected set of instances ($N = 20, 25, 30, 35$, and 40) exhibiting the longest classical runtimes. In this appendix, we present results for the

same instances (up to $N = 30$) using our universal parameter scaling obtained from Fig. 16. Note, however, that the normalization of QUBO matrices (such that $\max_{i \neq j} Q_{ij} = 2$) employed in Ref. [18] leads to a corresponding shift of Δ_γ values for each instance, see the blue dots in Fig. 16, which changes the fit parameters to $a = 5.945$ and $b = -0.800$ with respect to the values displayed in Fig. 16(a).

Thereby, we obtain the following LR-QAOA parameters: $\Delta_\gamma = 0.542$, $\Delta_\beta = 0.608$, $p = 30$ for $N = 20$, $\Delta_\gamma = 0.453$, $\Delta_\beta = 0.581$, $p = 47$ for $N = 25$, and $\Delta_\gamma = 0.392$, $\Delta_\beta = 0.560$, $p = 67$ for $N = 30$. In contrast, the values $\Delta_\gamma = 0.4$, $\Delta_\beta = 0.4$, $p = 100$ for $N = 20$, $\Delta_\gamma = 0.5$, $\Delta_\beta = 0.4$, $p = 50$ for $N = 25$, and $\Delta_\gamma = 0.4$, $\Delta_\beta = 0.5$, $p = 100$ for $N = 30$ are used in Ref. [18]. In Fig. 18, we see

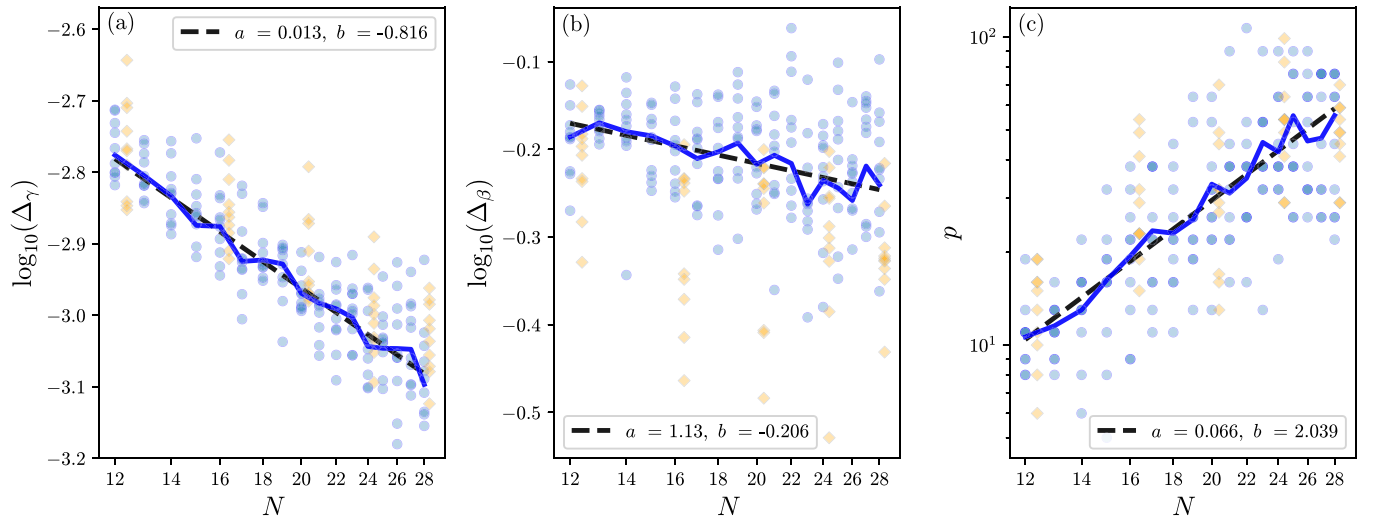


FIG. 16. MaxCut: optimized linear-ramp parameters (a) Δ_γ , (b) Δ_β , and (c) QAOA depth p as a function of number of qubits N on a log-log scale. The black dashed lines display algebraic fits $f(N) = aN^b$ to the $1/D$ -weighted geometric means (blue lines) and determine the universal parameter scaling used in Fig. 11(c).

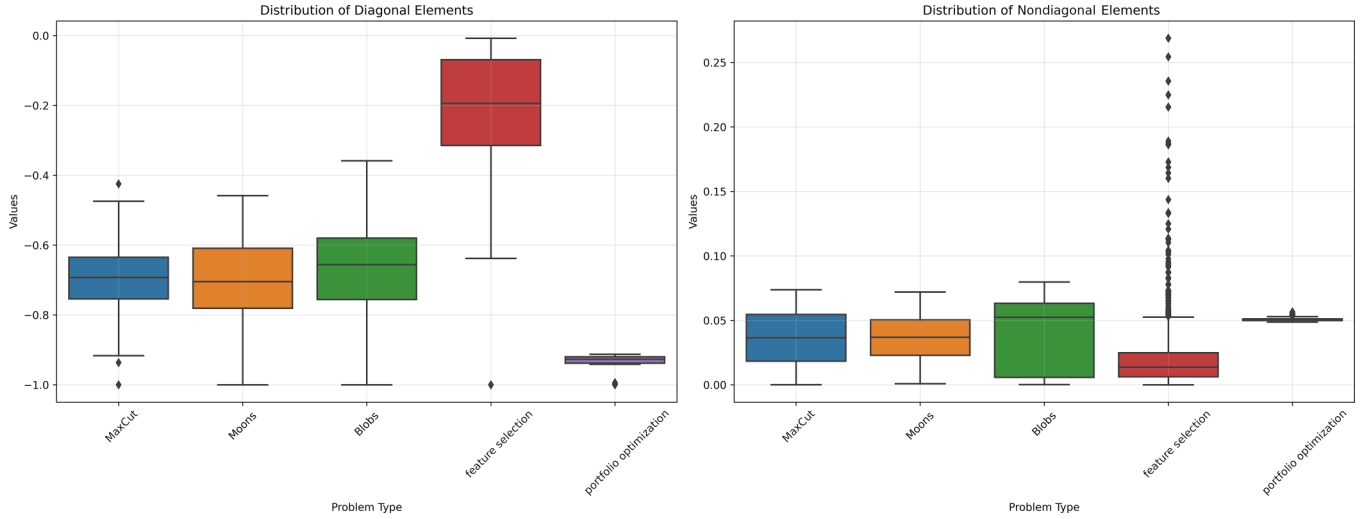


FIG. 17. Comparison of box plots representing the distribution of QUBO matrix elements between all problem types under consideration for problem size $N = 20$. The distributions of the QUBO matrices' diagonal elements are shown in the left panel; the distribution of the nondiagonal elements is shown in the right panel. The QUBO matrices have been normalized by the following factors: 7.39×10^{-5} (MaxCut), 0.0102 (clustering and Moons), 0.0112 (clustering and Blobs), 3.45 (feature selection), and 0.0215 (portfolio optimization).

that these two sets of parameters achieve (at least on average) almost identical results, with scaling coefficients $\alpha = 0.103$ and 0.100, respectively.

Comparing the values of D in Figs. 11(d) and 18, we see that the instances, which have been selected due to

their long classical runtimes, are also more difficult to solve for LR-QAOA, leading to larger D . Moreover, this increase of D appears to be more strongly pronounced for smaller problem sizes, which, in total amounts to a smaller scaling coefficient α .

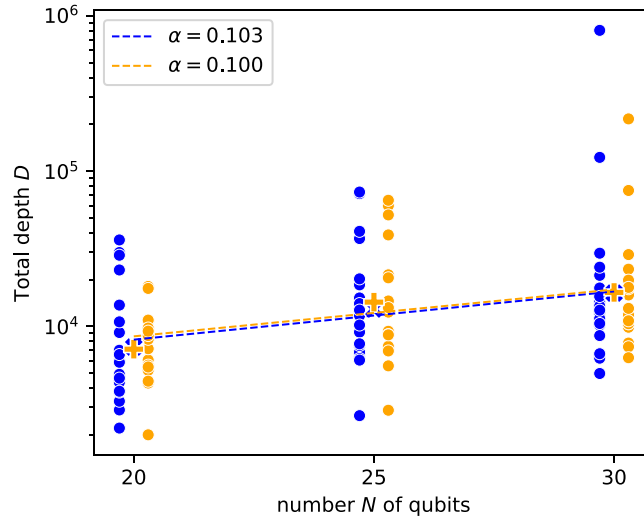


FIG. 18. Scaling of quantum runtime expressed as total depth D , see Eq. (16), for the set of selected, classically hard weighted MaxCut instances ($N = 20, 25, 30$) considered in Ref. [18]. The blue dots (slightly shifted to the left for better visibility) show the depths obtained with the LR-QAOA circuit parameters used in Ref. [18], whereas the orange dots (shifted to the right) display the results of our universal parameter scaling ansatz. The blue and orange crosses indicate the (geometric) mean values, respectively. The dashed lines represent exponential fits $D \propto 2^{\alpha N}$, with almost identical scaling coefficients ($\alpha = 0.103$ and 0.100)—similar to the one ($\alpha = 0.107$) found in Ref. [18] for problem sizes up to $N = 40$.

- [1] E. Farhi, J. Goldstone, and S. Gutmann, A quantum approximate optimization algorithm, [arXiv:1411.4028](#).
- [2] E. Farhi, J. Goldstone, S. Gutmann, and L. Zhou, The quantum approximate optimization algorithm and the Sherrington-Kirkpatrick model at infinite size, [Quantum](#) **6**, 759 (2022).
- [3] C. Grange, M. Poss, and E. Bourreau, An introduction to variational quantum algorithms for combinatorial optimization problems, [Ann. Oper. Res.](#) **21**, 363 (2023).
- [4] K. Blekos, D. Brand, A. Ceschini, C.-H. Chou, R.-H. Li, K. Pandya, and A. Summer, A review on quantum approximate optimization algorithm and its variants, [Phys. Rep.](#) **1068**, 1 (2024).
- [5] R. Shaydulin and Y. Alexeev, Evaluating quantum approximate optimization algorithm: A case study, in *2019 Tenth International Green and Sustainable Computing Conference (IGSC)* (IEEE, Alexandria, VA, 2019), pp. 1–6.
- [6] S. H. Sureshbabu, D. Herman, R. Shaydulin, J. Basso, S. Chakrabarti, Y. Sun, and M. Pistoia, Parameter setting in quantum approximate optimization of weighted problems, [Quantum](#) **8**, 1231 (2024).
- [7] E. Farhi, S. Gutmann, D. Ranard, and B. Villalonga, Lower bounding the maxcut of high girth 3-regular graphs using the QAOA, [arXiv:2503.12789](#).
- [8] V. Kremenetski, T. Hogg, S. Hadfield, S. J. Cotton, and N. M. Tubman, Quantum alternating operator ansatz (QAOA) phase diagrams and applications for quantum chemistry, [arXiv:2108.13056](#).
- [9] J. Sud, S. Hadfield, E. Rieffel, N. Tubman, and T. Hogg, Parameter-setting heuristic for the quantum alternating operator ansatz, [Phys. Rev. Res.](#) **6**, 023171 (2024).
- [10] F. G. S. L. Brandao, M. Broughton, E. Farhi, S. Gutmann, and H. Neven, For fixed control parameters the quantum approximate optimization algorithm’s objective function value concentrates for typical instances, [arXiv:1812.04170](#).
- [11] 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).
- [12] S. H. Sack and M. Serbyn, Quantum annealing initialization of the quantum approximate optimization algorithm, [Quantum](#) **5**, 491 (2021).
- [13] V. Kremenetski, A. Apte, T. Hogg, S. Hadfield, and N. M. Tubman, Quantum alternating operator ansatz (QAOA) beyond low depth with gradually changing unitaries, [arXiv:2305.04455](#).
- [14] R. Sakai, H. Matsuyama, W.-H. Tam, Y. Yamashiro, and K. Fujii, Linearly simplified QAOA parameters and transferability, [arXiv:2405.00655](#).
- [15] R. Shaydulin, S. Hadfield, T. Hogg, and I. Safro, Classical symmetries and the quantum approximate optimization algorithm, [Quantum Inf. Process.](#) **20**, 359 (2021).
- [16] G. B. Mbeng, R. Fazio, and G. Santoro, Quantum annealing: A journey through digitalization, control, and hybrid quantum variational schemes, [arXiv:1906.08948](#).
- [17] G. E. Crooks, Performance of the quantum approximate optimization algorithm on the maximum cut problem, [arXiv:1811.08419](#).
- [18] J. A. Montanez-Barrera and K. Michielsen, Towards a universal QAOA protocol: Evidence of a scaling advantage in solving some combinatorial optimization problems, [npj Quantum Inf.](#) **11**, 131 (2025).
- [19] S. Boulebnane and A. Montanaro, Solving Boolean satisfiability problems with the quantum approximate optimization algorithm, [PRX Quantum](#) **5**, 030348 (2024).
- [20] R. Shaydulin, C. Li, S. Chakrabarti, M. DeCross, D. Herman, N. Kumar, J. Larson, D. Lykov, P. Minssen, Y. Sun, Y. Alexeev, J. M. Dreiling, J. P. Gaebler, T. M. Gatterman, J. A. Gerber, K. Gilmore, D. Gresh, N. Hewitt, C. V. Horst, S. Hu, *et al.*, Evidence of scaling advantage for the quantum approximate optimization algorithm on a classically intractable problem, [Sci. Adv.](#) **10**, eadm6761 (2024).
- [21] F. Glover, G. Kochenberger, and Y. Du, A tutorial on formulating and using QUBO models, [arXiv:1811.11538](#).
- [22] H. Markowitz, Portfolio selection, [J. Finance](#) **7**, 77 (1952).
- [23] S. Brandhofer, D. Braun, V. Dehn, G. Hellstern, M. Hüls, Y. Ji, I. Polian, A. S. Bhatia, and T. Wellens, Benchmarking the performance of portfolio optimization with QAOA, [Quantum Inf. Process.](#) **22**, 25 (2022).
- [24] G. Hellstern, V. Dehn, and M. Zaefferer, Quantum computer based feature selection in machine learning, [IET Quantum Commun.](#) **5**, 232 (2024).
- [25] H. Hofmann, Statlog (German credit data), UCI Machine Learning Repository (1994), <https://doi.org/10.24432/C5NC77>.
- [26] S. Lloyd, Least squares quantization in pcm, [IEEE Trans. Inf. Theory](#) **28**, 129 (1982).
- [27] R. Sibson, SLINK: an optimally efficient algorithm for the single-link cluster method, [Comput. J.](#) **16**, 30 (1973).
- [28] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, *et al.*, A density-based algorithm for discovering clusters in large spatial databases with noise, in *KDD’96: Proceedings of the Second International Conference on Knowledge Discovery and Data Mining* (AAAI Press, Portland, OR, 1996), Vol. 96, pp. 226–231.
- [29] J. Saiphet, S. Suwanna, T. Chotibut, and A. Chantasri, Quantum approximate optimization and k-means algorithms for data clustering, [J. Phys.: Conf. Ser.](#) **1719**, 012100 (2021).
- [30] J. S. Otterbach, R. Manenti, N. Alidoust, A. Bestwick, M. Block, B. Bloom, S. Caldwell, N. Didier, E. S. Fried, S. Hong, P. Karalekas, C. B. Osborn, A. Papageorge, E. C. Peterson, G. Prawiroatmodjo, N. Rubin, C. A. Ryan, D. Scarabelli, M. Scheer, E. A. Sete, *et al.*, Unsupervised machine learning on a hybrid quantum computer, [arXiv:1712.05771](#).
- [31] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, SCIKIT-LEARN: Machine learning in Python, [J. Mach. Learn. Res.](#) **12**, 2825 (2011).
- [32] IBM ILOG CPLEX, V12.1: User’s manual for CPLEX, IBM ILOG CPLEX V12.1 (2009), https://public.dhe.ibm.com/software/websphere/ilog/docs/optimization/cplex/ps_usrmanplex.pdf.
- [33] Gurobi Optimization, LLC, Gurobi optimizer reference manual (2023), <https://www.gurobi.com/>.
- [34] I. Dunning, S. Gupta, and J. Silberholz, What works best when? a systematic evaluation of heuristics for max-cut and QUBO, [INFORMS J. Comput.](#) **30**, 608 (2018).
- [35] M. X. Goemans and D. P. Williamson, Improved approximation algorithms for maximum cut and satisfiability problems using

- semidefinite programming, *J. Assoc. Comput. Mach.* **42**, 1115 (1995).
- [36] QISKIT contributors, QISKIT: An open-source framework for quantum computing, Zenodo (2023), <https://doi.org/10.5281/zenodo.8190968>.
- [37] Z. Zhu, A. J. Ochoa, and H. G. Katzgraber, Fair sampling of ground-state configurations of binary optimization problems, *Phys. Rev. E* **99**, 063314 (2019).
- [38] M. Born and V. Fock, Beweis des Adiabatsatzes, *Z. Phys.* **51**, 165 (1928).
- [39] A. Dranov, J. Kellendonk, and R. Seiler, Discrete time adiabatic theorems for quantum mechanical systems, *J. Math. Phys.* **39**, 1340 (1998).
- [40] T. F. Rønnow, Z. Wang, J. Job, S. Boixo, S. V. Isakov, D. Wecker, J. M. Martinis, D. A. Lidar, and M. Troyer, Defining and detecting quantum speedup, *Science* **345**, 420 (2014).
- [41] T. Wellens, V. Dehn, G. Hellstern, K. Jayadevan, F. Reiter, and T. Wellens, Extrapolation method to optimize linear-ramp quantum approximate optimization algorithm parameters: dataset [Data set], Zenodo (2026), <https://doi.org/10.5281/zenodo.18302993>.