Editors' Suggestion

Monte Carlo graph search for quantum circuit optimization

Bodo Rosenhahn
Institute for Information Processing (tnt/L3S), Leibniz Universität Hannover, Hannover 30167, Germany

Tobias J. Osborne
Institute of Theoretical Physics and L3S, Leibniz Universität Hannover, Appelstrasse 2, 30167 Hannover, Germany

(Received 23 August 2023; accepted 17 November 2023; published 19 December 2023)

The building blocks of quantum algorithms and software are quantum gates, with the appropriate combination of quantum gates leading to a desired quantum circuit. Deep expert knowledge is necessary to discover effective combinations of quantum gates to achieve a desired quantum algorithm for solving a specific task. This is especially challenging for quantum machine learning and signal processing. For example, it is not trivial to design a quantum Fourier transform from scratch. This work proposes a quantum architecture search algorithm which is based on a Monte Carlo graph search and measures of importance sampling. It is applicable to the optimization of gate order for both discrete gates and gates containing continuous variables. Several numerical experiments demonstrate the applicability of the proposed method for the automatic discovery of quantum circuits.

DOI: 10.1103/PhysRevA.108.062615

I. INTRODUCTION

Quantum computing has brought about a paradigm shift in information processing and promises breakthroughs in the solution of industrial use cases [1–3], physics [4,5], medicine [6], chemistry [7], biology [8], robotics [9], general pattern recognition [10], machine learning [11–13], and much more. These opportunities are tempered, however, by the significant challenges arising in the discovery and application of quantum software. This is because the design of quantum algorithms still requires expert knowledge. Further, since quantum devices are still small and costly, it is essential to optimize resources, in particular, gate counts. Thus, the discovery and optimization of quantum circuits are of significant near-term relevance.

Methods for the automated search for optimal quantum circuits have been investigated in the literature, and the term *quantum architecture search* (QAS) has been adopted to describe this body of research. The name is borrowed and adapted from *neural architecture search* [14,15], which is devoted to the study and hyperparameter tuning of neural networks. Recent works on QAS are often specific to a problem setup; for example, it has been applied to quantum circuit structure learning [16] for finding the ground states of lithium hydride and the Heisenberg model in simulation, as well as for finding the ground state of a hydrogen gas. Many QAS variants are focused on discrete optimization and exploit optimization strategies for nondifferentiable optimization criteria. Here variants of Gibbs sampling [17], evolutional approaches [18], genetic algorithms [19,20], neural-network-based predictors [21], variants with noise-aware circuit learning [22], and the optimization of approximate solutions [23] have been suggested. A recent survey on QAS can be found in [24]. Going beyond discrete optimization, it is also possible to exploit gradient-descent-based optimization schemes [25,26] or reinforcement learning [27] for QAS.

A recent work [28] proposed a Monte Carlo tree search (MCTS) based on a multiarmed-bandit formulation. That paper is closest in spirit to our present work, and the promising results and challenges identified there inform and inspire our investigation. Traditional tree search does not share information between different trajectories, although an identical state may occur. Therefore, a MCTS algorithm does not merge duplicate states. Thus, if a state x can be reached via two different trajectories, it will be represented two times in the look-ahead tree. Thus, the rollout process (sometimes called *simulation*) can generate nodes and branches which are already part of the tree, leading to multiple identical circuits in the search. Since quantum gates act locally, many of them commute, automatically leading to circles in a quantum circuit graph (see Fig. 2 below for an illustration).

We overcome the challenges presented by cycles in the quantum circuit graph by proposing the use of a Monte Carlo graph search (MCGS) [29] to optimize a combination of mixed discrete and continuous variables. This is possible since most gates containing continuous variables are smooth (e.g., consisting of rotation coefficients), and thus, it is possible to automatically extract the Jacobians for a fast gradient descent while optimizing the quantum computation graph. Our contributions can be summarized as follows:

(1) We propose a Monte Carlo graph search algorithm for quantum architecture optimization.

(2) Our model allows for a joint discrete and continuous optimization of the quantum gate ordering and parameters.

(3) Several applications demonstrate its applicability, e.g., for the optimization of the quantum Fourier transform, diverse quantum cellular automatons, and simple quantum machine learning tasks.

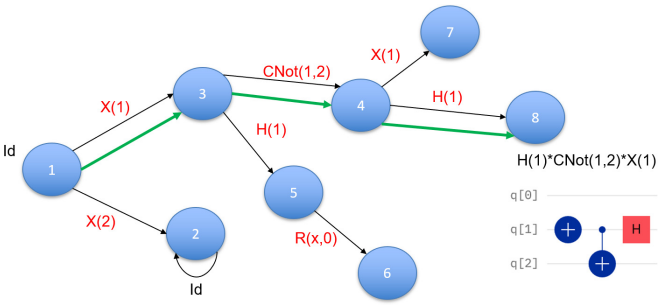


FIG. 1. Example graph of quantum circuits. The edges are labeled by possible elementary gates. The vertices are identified with the unitary operator built by taking the product of the gates along the shortest path. A trajectory on this graph directly corresponds to a quantum circuit.

II. PRELIMINARIES

In this section we give a brief overview of the physical systems we discuss in this paper and provide a description of the architecture search optimization strategies that we compare and contrast. In particular, reference methods frequently used for discrete optimization are briefly introduced. They are later used for a direct comparison with our proposed MCGS algorithm.

We focus on the setting where our quantum information processing device comprises a set of N *logical* qubits, arranged as a quantum register (see, e.g., [30,31] for further details). Thus, the Hilbert space of our system is furnished by $\mathcal{H} \equiv (\mathbb{C}^2)^{\otimes N} \cong \mathbb{C}^{2^N}$. In this way, e.g., a quantum state vector of a five-qubit register is a unit vector in $\mathbb{C}^{32}$. We assume throughout that the system is not subject to decoherence and remains pure.

Quantum gates are the basic building blocks of quantum circuits, similar to logic gates in digital circuits [32]. According to the axioms of quantum mechanics, quantum logic gates are represented by unitary matrices so that a gate acting on N qubits is represented by a $2^N \times 2^N$ unitary matrix, and the set of all such gates together with the group operation of matrix multiplication furnishes the symmetry group $\mathrm{U}(2^N)$. In order to describe explicit matrix representations we exploit the *computational basis* $\{|x_1, x_2, \ldots, x_N\rangle \,|\, x_j \in \{0, 1\}, j = 1, 2, \ldots, N\}$ furnished by the eigenstates of the Pauli Z operator on each qubit j.

Standard quantum gates include the Pauli (X, Y, Z) operations, as well as Hadamard, controlled NOT (CNOT), SWAP, phase-shift, and Toffoli gates, all of which are expressible as standardized unitary matrices with respect to the computational basis. The action of a quantum gate is extended to a register of any size exploiting the tensor product operation in the standard way. Most gates do not involve additional variables; however, a phase-shift gate $R_X(\theta)$ applies a complex rotation and involves the rotation angle θ as a free parameter. This parameter should then be jointly optimized together with the architecture of an overall quantum circuit.

A quantum circuit of length L is then described by an ordered tuple $(O(1), O(2), \ldots, O(L))$ of quantum gates; the resulting unitary operation U implemented by the circuit is the product

$$U = O(L)O(L-1)\cdots O(1). \tag{1}$$

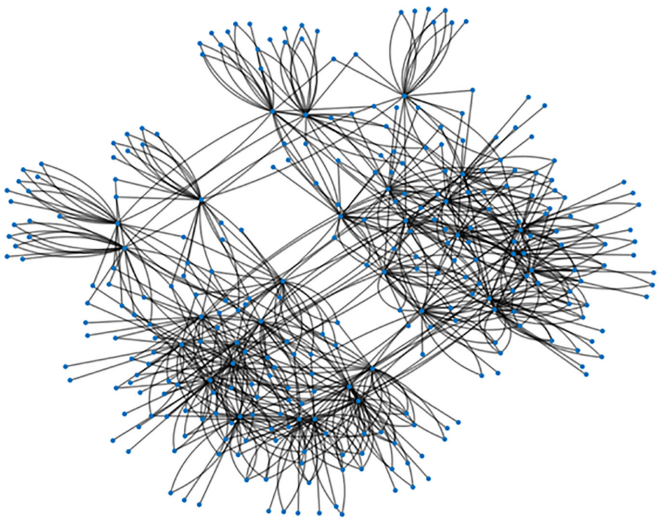


FIG. 2. Example graph of quantum circuits. The purpose of this visualization is to demonstrate the high degree of connectivity and the presence of multiple cycles in the generated quantum circuit graph. A large number of the cycles emerge from commuting local quantum gates. As products of gates are taken to build the unitary matrix representing the quantum circuit, one encounters a graph structure which is rapidly branching and merging. This branching and cycle structure indicates that the MCGS will likely supply superior performance relative to MCTS.

A. Genetic algorithms

A genetic algorithm (GA) belongs to the family of so-called evolutionary algorithms. A GA is a population-based metaheuristic inspired by biological evolution. It comprises a fitness function to evaluate individuals of a population; a selection process (driven by the fitness scores) to decide which individuals are used for reproduction; and genetic operators, such as crossovers, and mutations to generate new individuals. These new individuals form a new generation which is further evaluated in an ongoing evolution. Genetic algorithms are commonly exploited in discrete optimization, and the interested reader is referred to [33] for further details.

For the numerical experiments carried out in this paper the fitness function is directly given by the optimization task (a loss or quality score). Each individual (quantum circuit) I_1 is represented by an ordered tuple of quantum gates, e.g., $I_1 = (O_1(1), \ldots, O_1(n))$. For a crossover between two circuits I_1 and I_2, a point on both parents' chromosomes is randomly picked which is called the *crossover point*. Gates to the right of that point are swapped between the two parent chromosomes. This results in two children,

$$(O_1(1), \ldots, O_1(j), O_2(j+1), \ldots, O_2(n))$$

and

$$(O_2(1), \ldots, O_2(j), O_1(j+1), \ldots, O_1(n)),$$

each carrying some genetic information from both parents. A mutation is then furnished by a random exchange of a

quantum gate. A recent work on an evolutionary quantum architecture search for parametrized quantum circuits was presented in [34].

B. Particle filter

Particle filtering uses a set of samples (which are then called particles) to model a posterior distribution of a stochastic process given some observations. A particle filter is also called a *sequential Monte Carlo method* [35]. These Monte Carlo algorithms are commonly used to find approximate solutions for filtering problems of nonlinear state-space systems. More recently, they have been applied to quantum systems as quantum Monte Carlo methods [36]. In the controls literature, particle filters are exploited to estimate the posterior distribution of the state x_t of a dynamical system at time t conditioned on the data $p(x_t|z^t, u^t)$. This posterior is estimated via the following recursive formula:

$$p(x_t|z^t, u^t) = \eta_t \; p(z_t|x_t) \int p(x_t|u_t, x_{t-1}) \times \; p(x_{t-1}|z^{t-1}, u^{t-1}) \, dx_{t-1}, \tag{2}$$

where η_t is a normalization constant.

Three probability distributions are required for such a particle filter: (1) a so-called *measurement model*, $p(z_t|x_t)$, which gives the probability of measuring z_t when the system is in state x_t; (2) a *control model*, $p(x_t|u_t, x_{t-1})$, which models the effect of a control u_t on the system state and provides the probability that the system is in state x_t after executing control u_t at state x_{t-1}; and (3) an *initial state distribution* $p(x_0)$, which is required to specify the user's knowledge about the initial system state (see also [37]). In computer vision, the so-called *condensation algorithm* is a well-known example of how to perform a conditional density propagation for visual tracking [38].

The implementation of a particle filter can be very similar to that of a genetic algorithm. It can be based on M independent random variables ξ_0^i $(i = 1, \ldots, M)$ with a probability density $p(x_0)$. Based on the underlying distribution, e.g., representing a fitness score, M of these variables are selected, $\xi_k^i \to \hat{\xi}_k^i$, and diffused using a mutationlike operation, yielding a new set ξ_{k+1}^i.

C. Simulated annealing

Simulated annealing (SA) is another probabilistic technique to approximate the optimum of a given function [39]. The name derives from annealing in metallurgy, in which the process involves heating and a controlled cooling of a material to change and control its physical properties. As an optimization scheme the algorithm works iteratively with respect to time t given a state x_t. At each step, the simulated annealing heuristic samples a neighboring state $\hat{x}_t$ of the current state x_t. Then a probabilistic decision is made to decide whether to move to the new state $x_{t+1} = \hat{x}_t$ or to remain in the former state $x_{t+1} = x_t$. The probability of making the transition from the current state x_t to the new state $\hat{x}_t$ is defined by an acceptance probability function $P(e(x_t), e(\hat{x}_t), T)$. The function $e(x)$ evaluates the energy of this state, which is, in our case, the fitness score given by the optimization task (e.g., the ℓ_2 loss). The parameter T is a time-dependent variable dictating the behavior of the stochastic process according to a cooling scheme or annealing schedule. The P function is typically chosen in such a way that the probability of accepting an uphill move decreases with time and it decreases as the difference $e(\hat{x}_t) - e(x_t)$ increases. Thus, a small increase in error is likely to be accepted so that local minima can be avoided, whereas a larger error increase is not likely to be accepted. A typical function for P takes the form

$$P(e(x_t), e(\hat{x}_t), T) \propto \exp\left(-\frac{e(\hat{x}_t) - e(x_t)}{kT}\right), \tag{3}$$

with k being a *damping factor* $k > 0$.

D. Monte Carlo tree search

MCTS is a heuristic search algorithm for decision processes [29]. It makes use of random sampling and very efficiently balances the well-known exploration-exploitation dilemma in large search spaces. A typical example is provided by game states in which nonpromising game configurations are avoided, e.g., typical for board games such as chess and tic-tac-toe. MCTS is a common approach in reinforcement learning, typically in combination with deep reinforcement learning [40]. As it visits more interesting nodes more frequently, it grows asymmetrically and focuses the search time on more relevant parts of the tree.

Saffidine *et al.* [41] presented a framework for testing various algorithms that deal with transpositions in MCTS. They called this framework the upper confidence bound for Direct acyclic graphs and applied this formalism to overcome the exploration-exploitation dilemma. Their search strategy in the directed acyclic graph (DAG) follows the upper confidence bounds for trees algorithm [42]. These predecessors were more recently applied to Monte Carlo graph search to optimize game play in AlphaZero-based reinforcement learning [43].

III. PROBABILISTIC GRAPHICAL MODELS

In this section we describe the graphical model we exploit to characterize the search space of quantum circuits.

We assume throughout that we have a fixed set $\mathcal{OP} = \{O_1, O_2, \ldots\}$ of elementary quantum gates that we are allowed to apply. Note that in $\mathcal{OP}$ the same unitary gate acting on different qubits is considered to be a different elementary gate. For example, the Pauli X operator acting on qubit 1, written here as $X(1)$, is considered to be a different elementary gate from $X(2)$, which is the Pauli X operator acting on qubit 2. Starting with the identity operator $\mathbb{I}$, we can build quantum circuits by selecting elementary gates from $\mathcal{OP}$ and multiplying from the left. (Thanks to the universality theorem [31], we know that we can approximate an arbitrary unitary to arbitrarily good accuracy with a sufficiently long product of such gates.)

We associate a vertex from a vertex set V with each quantum circuit built from a product of elementary gates from $\mathcal{OP}$. We connect with an edge two such vertices if

```
MCGS Algorithm (requirements)
G=(V,E,Q) %Parameters and operators for the Graph (Vertices, Edges and Quality scores on Nodes)
OP        %Operator pool to sample from
T         % Target unitary matrix
Procedure [G indu]=MCGS(OP,T)
  G.V(1)=I                 %start quantum code with unit matrix at first node
  E={}                     % start with no edges
  Q(1)=evaluate(G.V(1),T)   % evaluate the first node with respect to task T
  tasksolved=0
  while (tasksolved==0)
    [v, indv]=sampleNode(G)        %sample node v and its index in Graph based on Q
    o=sampleOperator(G,indv,OP)    % sample available Operator
    u = o*v                        %generate unitary to test
     if isel(u, G.V)                          % Does the unitary already exist in graph ?
                 indu=find(u,G)               % find index
                 addedge(G,indv,indu,o)       % add edge from v to the existing node with the operator o
     else
                  indu=addnode(G,u)           % add new node to graph with a new index
                  addedge(G,indv,indu,o)      % add edge from v to the new node u with the operator o
                  update(Q)                   %update the Quality scores
     end
     if(u==T) tasksolved=1; end   % if task is solved, end the loop, return the Graph and the index of the Solution
   end
   return [G indu]
end
```

FIG. 3. Basic steps of the MCGS quantum circuit optimization algorithm. The currently implemented gates and their invocations are tabulated in the Appendix.

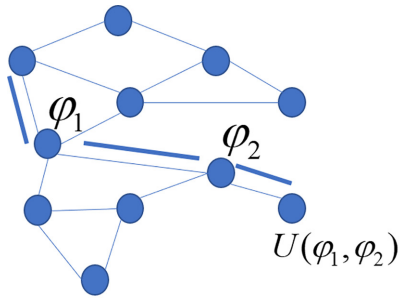
the corresponding quantum circuit differs (on the left) by an elementary gate. In this way, a collection of quantum circuits is endowed with a graph structure, with vertices decorated by quantum circuits and edges weighted by elementary gates. In Fig. 1 a tiny example graph for differently ordered quantum gates is depicted. The edges are labeled by possible gates of the quantum circuit. The nodes are decorated by the resulting unitary when concatenating the operations along the shortest path. Thus, each node is identified with a possible quantum circuit. It is important to note that this graph contains cycles since identical quantum circuits have multiple representations with different gates and gate orders.

In Fig. 2 we illustrate a few steps of such a growing graph model. As the depth of the graph grows exponentially with the number of gates and nodes, it is computationally infeasible to precompute such a graph for all possible circuits. For example, a tiny set of 20 elementary quantum gates can be assembled to build 20^5 combinations for quantum circuits of length 5. Thus, it is not possible to evaluate all configurations in a feasible time to solve for a specific optimization task. Figure 2 illustrates that the resulting graph contains many cycles, which in return justifies the use of a MCGS approach.

Since the graph model generated by the evaluation of quantum gates can have cycles, we apply a Monte Carlo search on the graph model with quantum gates as transitions. Thus, given a specific task, every node will receive a quality score which is used to compute a probability for the selection of this node. Based on the random selection and already explored operations, a new operation is randomly selected to grow the graph. Once a solution is found, an efficient quantum circuit can be generated by computing the shortest path in the graph from the start node to the target node.

This strategy is formalized as follows: We have a set of vertices V, associated with quantum circuits, of a graph $G = (V; E)$ with $V = \{v_1, \ldots, v_n\}$, and we build a probability function $p(v_i)$, $\sum_i p(v_i) = 1$, which assigns to each node of the graph a probability for selection. Poisson sampling is then exploited as the underlying sampling process. It is assumed that each vertex of the graph is an independent Bernoulli trial. Following standard mathematical conventions, the first-order inclusion probability of the ith element of the graph is denoted by the symbol $\pi_i = p(v_i)$. We further associate with each vertex of the graph an underlying task specific *quality score* $s_i \geqslant 0$. (Here larger values of s_i imply better quality.) Accordingly, we compute the first-order inclusion probability via $\pi_i = \frac{s_i}{\sum_j s_j}$. This paradigm of Monte Carlo search [44] and adapted Gibbs sampling [45] is used to iteratively grow a graph containing the effects of ordered quantum operations as trajectories in this graph (see Fig. 1).

Our MCGS procedure is now as follows: Given a set of vertices $V = \{v_1, \ldots, v_n\}$ and a probability function $p(v_i)$, Poisson sampling is used to select an existing node v_i from the graph. A quantum gate is then randomly selected from the set $\mathcal{OP}$ of elementary gates according to a uniform distribution and applied (from the left) to the quantum circuit associated with v_i. The result is a new circuit, which is associated with either an existing node or a new one [46]. If the circuit at a node already exists, e.g., v_j, a new edge (v_i, v_j) is added to the graph, decorated with edge weight given by the applied elementary gate. If the quantum circuit does not exist, the graph is extended by adding a new node v_{N+1} and an edge (v_i, v_{N+1}). Since the probabilities vary with respect to the



a) Find shortest Path in Graph
b) Collect continuous variables (denoted with phi)
c) Compute unitary U in unknowns phi

$$U(\varphi_1,\varphi_2) = U(4,\varphi_2)U(3)\ U(2,\varphi_1)\ U(1)$$

d) Compute Loss Fuction

$$L(\varphi_1,\varphi_2) = \left\| O - U(\varphi_1,\varphi_2) \right\|_F$$

e) Minimize loss using Gradient descent
and return the Quality

FIG. 4. Simple example to visualize the basic steps for the optimization of quantum circuits involving continuous variables.

quality score of the nodes, the graph grows asymmetrically. Once a node is reached which (sufficiently accurately) solves the optimization task, the shortest path from v_1 to the target node gives the shortest available quantum circuit (see also Fig. 1). The basic steps of the MCGS algorithm are summarized in Fig. 3.

Note that such a graph can also be reused for different kinds of optimizations, and it can be analyzed very generally to identify cycles, clusters, and other structural properties of the effect of quantum gates.

Optimization of continuous variables

Several gates can contain continuous variables for optimization, e.g., phase-shift gates,

$$P(\phi) = \begin{pmatrix} 1 & 0 \\ 0 & \exp(i\phi) \end{pmatrix}, \tag{4}$$

with the corresponding Jacobians

$$\frac{\partial P(\phi)}{\partial \phi} = \begin{pmatrix} 0 & 0 \\ 0 & i\exp(i\phi) \end{pmatrix}. \tag{5}$$

Since the involved functions are smooth and differentiable, the Jacobian of such a matrix and the Jacobian for a chain of operations are easy to compute via the product rule. Thus, given a quantum circuit of length L which is described by an ordered tuple $(O(1), O(2), \ldots, O(L))$ of quantum gates and $\Phi = (\phi_1 \ldots \phi_k)$ continuous variables within this chain, the resulting unitary operation is then a function $U(\Phi) = U(\phi_1, \ldots, \phi_k)$. This function is typically used for the optimization of a loss function, for example, the distance from a target matrix O, e.g., a density-functional-theory matrix. For numerical and implementation convenience the loss function was chosen to be the Frobenius norm between O and $U(\Phi)$,

$$\begin{aligned} L(\Phi) &= \|O - U(\Phi)\|_F \\ &= \sqrt{\mathrm{tr}\{[O - U(\Phi)]^\dagger [O - U(\Phi)]\}}. \end{aligned} \tag{6}$$

Although the Frobenius norm does not have a simple operational interpretation, it is easy to compute both numerically and also experimentally via a SWAP test.

The Jacobian of the loss function is given by

$$\nabla_\Phi L(\Phi) = \left[\frac{\partial L(\Phi)}{\partial \phi_1}, \ldots, \frac{\partial L(\Phi)}{\partial \phi_k} \right]. \tag{7}$$

This vector can be numerically computed and used for optimization by using automatic differentiation. Optimization of the involved parameters can be then carried out with a gradient-descent iteration using $\Phi^{t+1} = \Phi^t - \eta \nabla_\Phi L(\Phi)$. Here η denotes a damping factor which has been set to 0.2 for all our experiments. Figure 4 summarizes the basic steps for the optimization of continuous variables in the MCGS.

IV. EXPERIMENTS

Since the MCGS algorithm we describe here is capable of optimizing both discrete and mixed discrete and continuous settings, the experiments are divided into two parts. In the first part only results for discrete optimizations are presented. In the second part we also describe some experiments involving mixed discrete and continuous variables.

A. Discrete quantum architecture search

The first experiment we conducted targeted the optimization of the quantum Fourier transform. It is well known that the steps of the radix-2 fast Fourier transform (FFT) can be realized as a quantum circuit using primarily phase-shift and Hadamard gates. For example, for an eight-dimensional FFT, three qubits are sufficient, and seven gates can be used to compute the FFT matrix.

In the first experiment we compare the optimization of quantum circuits of length $1, 2, \ldots, 7$ for a given predefined set of elementary quantum gates. Note that for a database of elementary gates of size 32 and a circuit length of 7 there are $32^7 \sim 3.5 \times 10^{10}$ different quantum circuits possible; the search space grows exponentially and is already infeasibly large (for exhaustive searches) for larger circuits. In Fig. 5 we present the comparison of four different baselines based on a naive random sampling, a genetic algorithm (Sec. II A), a particle filter (Sec. II B), and simulated annealing (Sec. II C), along with our proposed Monte Carlo graph search. The x axis records the circuit length required for a predefined quantum circuit. The complexity of the optimization problem increasing exponentially with circuit length. Note that the y axis is scaled with the log function, so that a linear slope indicates an exponential growth in complexity. The error bars depict the standard deviation. For this experiment, the number of experiments or observations was set to 10. In comparison to the four

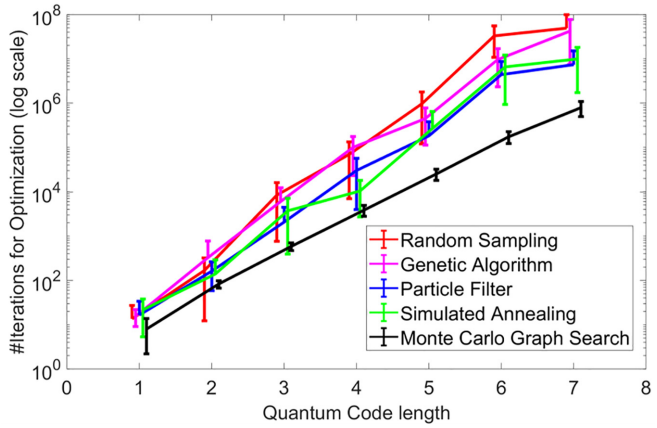


FIG. 5. Number of required samples for quantum circuit optimization for differing circuit lengths. Compared are random sampling, a genetic algorithm, a particle filter, simulated annealing, and the proposed MCGS model. The bars depict the standard deviation. The diagram is illustrated with respect to a logarithmic y-axis scale. One observes in this way that the MCGS already requires an order of magnitude fewer samples than the four baselines. (Quantities plotted are dimensionless.)

baselines, our proposed Monte Carlo graph search requires far fewer samples to reach a solution. One explanation for this is that unnecessary samples (e.g., leading to cycles, etc.) are efficiently avoided.

In the next experiment we analyze the efficiency of the generated quantum circuits in terms of the number of required gates. As the computation graph contains cycles, a valid question is whether standard sampling-based approaches for QAS can lead to solutions which require more gates than necessary, resulting in inefficient circuits. The approaches of Gibbs sampling, simulated annealing, and the proposed MCGS model allow one to optimize quantum circuits where the resulting code length is not fixed. Due to the diffusion and crossover steps, it is not clear how one can do this for the particle filter and the genetic algorithms, so these approaches were omitted for this experiment. In Fig. 6 the mean and standard deviation of the random sampling,

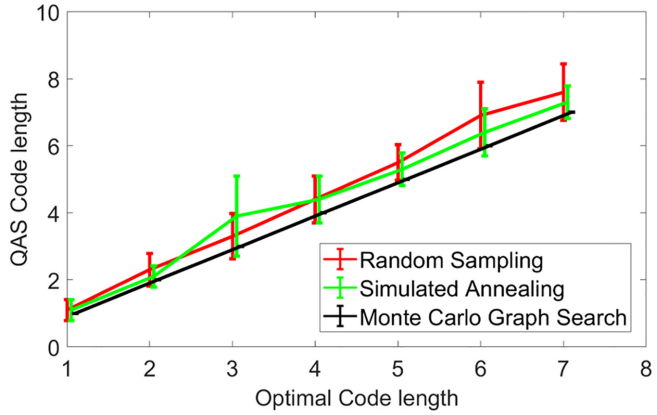


FIG. 6. Optimal circuit length versus the circuit length of different architecture search algorithms. (Quantities plotted are dimensionless.)

TABLE I. Rule set for automatons 30, 90, 110, and 184.

Rule	111	110	101	100	011	010	001	000
30 (=00011110)	0	0	0	1	1	1	1	0
90 (=01011010)	0	1	0	1	1	0	1	0
110 (=01101110)	0	1	1	0	1	1	1	0
184 (=10111000)	1	0	1	1	1	0	0	0

simulated annealing, and MCGS for varying optimal-circuit-length tasks is shown. The x axis is labeled by the optimal circuit length, and the y axis is labeled by the required circuit lengths (including the standard deviation) of the different optimizers. The optimal graph is a straight line, which was achieved by our proposed MCGS. Thus, the MCGS always finds the optimal length, whereas the sampling and annealing schemes more often tend to find inefficient solutions. Thus, the proposed MCGS jointly ensures efficient models during optimization.

B. Quantum circuits for classical cellular automatons

A cellular automaton (CA) is a mapping on a set of states of connected cells (e.g., arranged as a graph). Each cell has a cell state (e.g., binary) which changes according to a predefined set of rules given by the local neighbors of each cell. Simple and nontrivial cellular automatons are furnished already for one-dimensional graphs, with two possible states per cell. The neighbors are defined as the direct adjacent cells on either side of the cell. In this setting the rules for changing the state of each cell can be defined as a mapping from a three-dimensional binary state to a new binary state. Thus, there are $2^3 = 8$ patterns for a neighborhood. There are then $2^8 = 256$ possible combinations for rules which describe 256 different cellular automatons. In general they are referred to by their Wolfram code and are called R-X automatons, with X being a number between 0 and 255. Several papers have analyzed and compared these 256 cellular automatons. The cellular automatons defined by rule 30, rule 90, rule 110, and rule 184 are particularly interesting and are given by the mappings

Rule 30 exhibits so-called class-3 behavior. This means that even simple input patterns lead to chaotic, and seemingly random, histories. When starting from a single live cell, rule 90 produces a space-time diagram resembling the Sierpinski triangle. Rule 110, similar to the Game of Life, exhibits what is called class-4 behavior, which means it is neither completely random nor completely repetitive. Finally, rule 184 is notable for solving the majority problem as well as for its ability to simultaneously describe several seemingly quite different particle systems. For examples rule 184 can be used as a simple model for traffic flow. Thus, several of these transition rules exhibit interesting aspects for researchers in mathematics and optimization, as well as biology and physics. Due to the simple definition of these transition rules and their consequent rich behavior, these CAs provide a diverse and interesting class of targets for quantum circuits. The rule sets for cellular automatons 30, 90, 110 and 184 are summarized in Table I. In Fig. 7 we visualize the basic concept of the application of the MCGS. Given an eight-dimensional one-hot

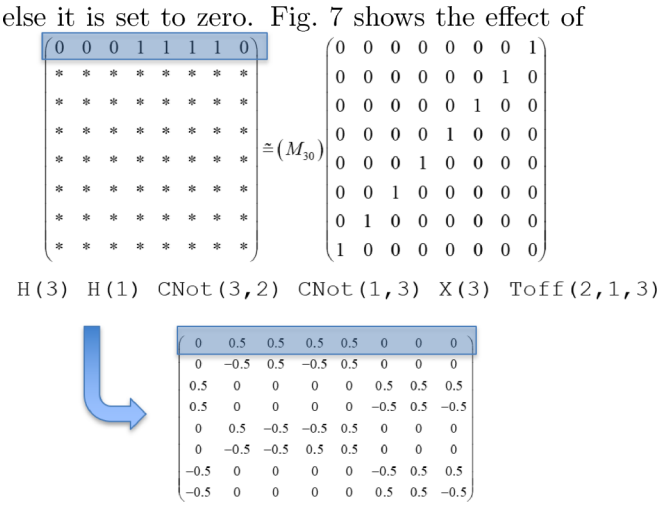


FIG. 7. Concept for a classical cellular automaton as a target for a quantum circuit. In this case the M_{30} automaton is targeted: The binary eight-dimensional input vector encodes the three-dimensional binary state using the one-hot encoding. A quantum circuit is searched for that, when applied to $|000\rangle$, produces the corresponding eight-dimensional vector according to the rule that if the probability is larger than zero (or a small threshold), the bit is set to 1 and is otherwise set to 0.

encoded input vector (encoding the three input binary elements), the quantum circuit is applied to the one-hot encoded vector. The qubit $|000\rangle$ is finally used for readout. If the probability is larger than zero (or above a small threshold), the bit is set to 1; otherwise, it is set to 0. Figure 7 shows the effect of all possible inputs (given as the reverse identity matrix) and the resulting value of the first qubit. The resulting row has to match the rule set of the cellular automatons, as in Table I.

We used our method to optimize quantum circuits for all possible eight-entry vectors corresponding to the 256 Wolfram Codes. The circuits for these optimized automatons are accessible as CSV files [47]. The computational effort to optimize the 256 CAs using the proposed methods are summarized in Table II. In accordance with the earlier experiment the proposed MCGS algorithm requires the fewest number of samples. Here it is also advantageous that the graph can be iteratively extended as the CA vectors are optimized.

C. Mixed discrete-continuous quantum architecture search

Our approach exploiting the Monte Carlo graph search can also be easily extended to optimize circuits involving a combination of discrete and continuous variables, as outlined before. In the following experiments we use quantum architectures to solve simple machine learning tasks.

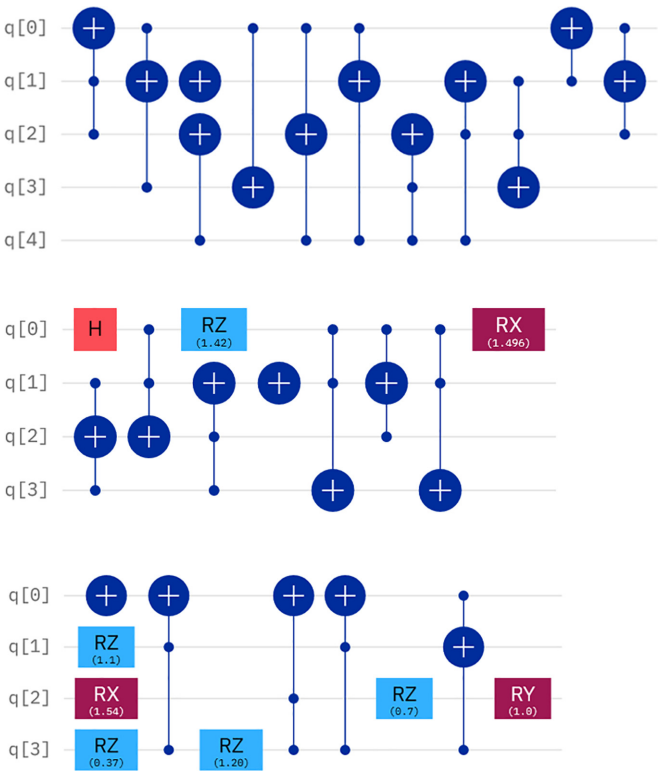


FIG. 8. Optimized quantum circuits for the wine, zoo, and iris datasets. (Images have been generated using QUANTUM COMPOSER [48].)

For the experiments, the classical *wine*, *zoo*, and *iris* datasets were used. The datasets present multicriterial classification tasks, with three categories for the wine dataset, seven categories for the zoo dataset, and three for the iris dataset. The datasets are all available at the University of California, Irvine, repository [49]. To model a classification task using a quantum circuit, first, the data are encoded as a higher-dimensional binary vector. Taking the iris dataset as a toy example, it consists of four-dimensional data encoding *sepal length*, *sepal width*, *petal length*, and *petal width*. After separating training and test data, a kMeans clustering on each dimension with $k = 3$ is used on the training data. Thus, every data point can be encoded in a $(4 \times 3 = 12)$-dimensional binary vector which contains exactly four nonzero entries. For the given cluster centers, the same can be done with the test data. Thus, a binary encoding is used to represent the datasets. Table III summarizes the datasets, the number of features (the dimension of each sample), its binary dimensionality, and the qubits used to represent the problem, as well as the amount of training data, the amount of target classes, and the gained accuracy with the optimized quantum model. Similar to Fig. 7,

TABLE II. The number of iterations and average circuit length for the optimization of all 256 CA Wolfram codes (lower values are better). RS = random sampling, GA = genetic algorithm, PF = particle filter, SA = simulated annealing, and MCGS = Monte Carlo graph search.

Method	RS	GA	PF	SA	MCGS
Iterations ↓	3.342.348	3.409.430	1.274.757	1.852.016	380.670
Average circuit length ↓	9.05	11	9.5	10.1	8.8

TABLE III. QML datasets: Overview and performance.

Dataset	Dim	BDim	Qubits	No. of training data	No. of classes	Accuracy
Iris	4	12	4	100	3	95%
Wine	13	26	5	133	3	90%
Zoo	16	16	4	75	7	92%

part of the quantum register is used to encode the probability of a classification label. The final decision is then based on the highest probability.

Figure 8 shows example outcomes of the optimized quantum codes for the wine and iris datasets. Note that the results can vary considerably. This depends on the random selection of training and test data and the random process of the graph generation. Table III summarizes the three datasets used and the overall performance. Note that the overall quality is similar to the results obtained with decision trees or shallow neural networks.

V. SUMMARY

In this paper we have proposed a quantum architecture search algorithm based on Monte Carlo graph search and measures of importance sampling. Each trajectory in this graph leads to a quantum circuit which can be evaluated according to whether it achieves a specific task. Our model also allows for the optimization of mixed discrete and continuous gates, and several experiments demonstrated the applicability for different tasks, such as matrix factorization, producing cellular automaton vectors, and simple machine learning models. A comparison with classical approaches such as greedy sampling, genetic algorithms, particle filter, and simulated annealing was carried out and demonstrated that the graph model performs more efficiently since cycles and redundancies are explicitly avoided. The shortest path from the start node to the target node provides efficient algorithms in terms of circuit length. The complexity of all investigated algorithms heavily increases with an increasing number of qubits and an increasing number of operators which can act on these qubits. We strongly believe that this makes

TABLE IV. List of (currently) implemented operators.

Name	Function call	Parameters	Matrix (simple base form)
Id	IdF(dim)	dim	$\begin{pmatrix}1 & 0\\ 0 & 1\end{pmatrix}$
Pauli X	XF(pos,dim)	position, dim	$\begin{pmatrix}0 & 1\\ 1 & 0\end{pmatrix}$
Pauli Z	ZF(pos,dim)	position, dim	$\begin{pmatrix}1 & 0\\ 0 & -1\end{pmatrix}$
Hadamard	HF(pos,dim)	position, dim	$\frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}$
T	TF(pos,dim)	position, dim	$\begin{pmatrix}1 & 0\\ 0 & \exp(\frac{-i\pi}{4})\end{pmatrix}$
Controlled T (CT)	CTF(pos,dim)	position, dim	$\begin{pmatrix}1 & 0\\ 0 & \exp(\frac{i\pi}{4})\end{pmatrix}$
Rotation around X axis (RotX)	RXF(pos, θ,dim)	position, angle, dim	$\begin{pmatrix}\cos(\theta) & -i\sin(\theta)\\ -i\sin(\theta) & \cos(\theta)\end{pmatrix}$
Rotation around Y axis (RotY)	RYF(pos, θ,dim)	position, angle, dim	$\begin{pmatrix}\cos(\theta) & -\sin(\theta)\\ \sin(\theta) & \cos(\theta)\end{pmatrix}$
Rotation around Z axis (RotZ)	RZZ(pos, θ,dim)	position, angle, dim	$\begin{pmatrix}\exp(-i\theta/2) & 0\\ 0 & \exp(i\theta/2)\end{pmatrix}$
Controlled NOT (CNOT)	CNotF(cb,b,dim)	control bit, controlled bit, dimension	$\begin{pmatrix}1&0&0&0\\0&1&0&0\\0&0&0&1\\0&0&1&0\end{pmatrix}$
Controlled U (CU)	CUF(cb,U,dim)	position, U matrix, dimension	$\begin{pmatrix}1&0&0&0\\0&1&0&0\\0&0&U(1,1)&U(1,2)\\0&0&U(2,1)&U(2,2)\end{pmatrix}$
SWAP	XXF(a,b,dim)	first and second bits to swap, dimension	$\begin{pmatrix}1&0&0&0\\0&0&1&0\\0&1&0&0\\0&0&0&1\end{pmatrix}$
Controlled Z	CZ(cb,b,dim)	control bit, controlled bit, dimension	$\begin{pmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&-1\end{pmatrix}$
Toffoli	ToffF(c1,c2,cb,dim)	two control bits, controlled bit, dimension	see the literature

highly efficient graph search algorithms unavoidable in the future. Another future challenge is to overcome the computation time required by the graph model as the number of nodes (and with that the number of required comparisons) is increased. One immediate and relevant next step is to generalize the method described here to apply to mixed states and completely positive maps. Future investigations will also cover the extension of the results presented here to the mixed-state case.

Our source code for optimization is publicly available from Leibniz Universität Hannover [47].

ACKNOWLEDGMENTS

This work was supported, in part, by Quantum Valley Lower Saxony and the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation), Project No. 274200144, SFB1227, under Germany's Excellence Strategies EXC-2123 QuantumFrontiers and EXC-2122 PhoenixD.

APPENDIX

The currently implemented gates and their invocations are given in Table IV.

[1] R. Rietsche, C. Dremel, S. Bosch, L. Steinacker, M. Meckel, and J.-M. Leimeister, Electron-Markets **32**, 2525 (2022).

[2] J. Preskill, Quantum **2**, 79 (2018).

[3] S. Endo, S. C. Benjamin, and Y. Li, Phys. Rev. X **8**, 031027 (2018).

[4] S. P. Jordan, K. S. Lee, and J. Preskill, Science **336**, 1130 (2012).

[5] P. Bargassa, T. Cabos, A. Cordeiro Oudot Choi, T. Hessel, and S. Cavinato, Phys. Rev. D **104**, 096004 (2021).

[6] J. Davids, N. Lidströmer, and H. Ashrafian, in *Artificial Intelligence in Medicine*, edited by N. Lidströmer and H. Ashrafian (Springer, Cham, 2022), pp. 423–446.

[7] Y. Cao, J. Romero, J. P. Olson, M. Degroote, P. D. Johnson, M. Kieferová, I. D. Kivlichan, T. Menke, B. Peropadre, N. P. D. Sawaya, S. Sim, L. Veis, and A. Aspuru-Guzik, Chem. Rev. **119**, 10856 (2019).

[8] V. Marx, Nat. Methods **18**, 715 (2021).

[9] M. Mannone, V. Seidita, and A. Chella, Swarm Evol. Comput. **79**, 101297 (2023).

[10] F. Bapst, W. Bhimji, P. Calafiura, H. Gray, W. Lavrijsen, and L. Linder, Comput. Software Big Sci. **4**, 1 (2020).

[11] A. Mott, J. Job, J.-R. Vlimant, D. Lidar, and M. Spiropulu, Nature (London) **550**, 375 (2017).

[12] S. L. Wu, J. Chan, W. Guan, S. Sun, A. Wang, C. Zhou, M. Livny, F. Carminati, A. Di Meglio, A. C. Y. Li, J. Lykken, P. Spentzouris, S. Y.-C. Chen, S. Yoo, and T.-C. Wei, J. Phys. G **48**, 125003 (2021).

[13] D. Willsch, M. Willsch, H. De Raedt, and K. Michielsen, Comput. Phys. Commun. **248**, 107006 (2020).

[14] R. Miikkulainen, in *Encyclopedia of Machine Learning and Data Science*, edited by D. Phung, G. I. Webb, and C. Sammut (Springer, New York, 2020), pp. 1–8.

[15] S. Xie, H. Zheng, C. Liu, and L. Lin, in *International Conference on Learning Representations* (2019).

[16] M. Ostaszewski, E. Grant, and M. Benedetti, Quantum **5**, 391 (2021).

[17] L. Li, M. Fan, M. Coram, P. Riley, and S. Leichenauer, Phys. Rev. Res. **2**, 023074 (2020).

[18] L. Franken, B. Georgiev, S. Mucke, M. Wolter, R. Heese, C. Bauckhage, and N. Piatkowski, in *2022 IEEE Congress on Evolutionary Computation (CEC)* (IEEE, Piscataway, NJ, 2022), pp. 1–8.

[19] R. Rasconi and A. Oddi, Proc. AAAI Conf. Artif. Intell. **33**, 7707 (2019).

[20] U. Las Heras, U. Alvarez-Rodriguez, E. Solano, and M. Sanz, Phys. Rev. Lett. **116**, 230504 (2016).

[21] S.-X. Zhang, C.-Y. Hsieh, S. Zhang, and H. Yao, Mach. Learn.: Sci. Technol. **2**, 045027 (2021).

[22] L. Cincio, K. Rudinger, M. Sarovar, and P. J. Coles, PRX Quantum **2**, 010324 (2021).

[23] L. Zhou, S.-T. Wang, S. Choi, H. Pichler, and M. D. Lukin, Phys. Rev. X **10**, 021067 (2020).

[24] W. Zhu, J. Pi, and Q. Peng, in *Proceedings of the 6th International Conference on Algorithms, Computing and Systems, ICACS '22* (Association for Computing Machinery, New York, 2023).

[25] M. Watabe, K. Shiba, C.-C. Chen, M. Sogabe, K. Sakamoto, and T. Sogabe, Quantum Rep. **3**, 333 (2021).

[26] S.-X. Zhang, C.-Y. Hsieh, S. Zhang, and H. Yao, Quantum Sci. Technol. **7**, 045023 (2022).

[27] M. M. Wauters, E. Panizon, G. B. Mbeng, and G. E. Santoro, Phys. Rev. Res. **2**, 033446 (2020).

[28] P. Wang, M. Usman, U. Parampalli, L. C. L. Hollenberg, and C. R. Myers, IEEE Trans. Quantum Eng. **4**, 1 (2023).

[29] M. C. Fu, in *Proceedings of the 2018 Winter Simulation Conference* (IEEE Press, Gothenburg, Sweden, 2018), pp. 222–236.

[30] P. Kaye, R. Laflamme, and M. Mosca, *An Introduction to Quantum Computing* (Oxford University Press, New York, 2007).

[31] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information* (Cambridge University Press, Cambridge, 2000).

[32] P. Selinger, Math. Struct. Comput. Sci. **14**, 527 (2004).

[33] A. E. Eiben and J. E. Smith, *Introduction to Evolutionary Computing* (Springer, New York, 2003).

[34] L. Ding and L. Spector, Entropy **25**, 93 (2023).

[35] A. G. Wills and T. B. Schön, Annu. Rev. Control, Rob., Auton. Syst. **6**, 159 (2023).

[36] J. Gubernatis, N. Kawashima, and P. Werner, *Quantum Monte Carlo Methods: Algorithms for Lattice Models* (Cambridge University Press, Cambridge, 2016).

[37] S. Thrun, J. Langford, and V. Verma, in *Advances in Neural Information Processing Systems*, edited by T. Dietterich, S. Becker, and Z. Ghahramani (MIT Press, Cambridge, MA, 2001), Vol. 14, pp. 961–968.

[38] A. Blake and M. Isard, in *Advances in Neural Information Processing Systems*, edited by M. Mozer, M. Jordan, and T. Petsche (MIT Press, Cambridge, MA, 1996), Vol. 9, pp. 361–365.

[39] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, Science **220**, 671 (1983).
[40] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, Nature (London) **529**, 484 (2016).
[41] A. Saffidine, T. Cazenave, and J. Méhat, Knowl.-Based Syst. **34**, 26 (2012).
[42] P. Auer, N. Cesa-Bianchi, and P. Fischer, Mach. Learn. **47**, 235 (2002).
[43] J. Czech, P. Korus, and K. Kersting, Proc. Int. Conf. Auto. Plann. Schedul. **31**, 103 (2021)
[44] N. Metropolis and S. Ulam, J. Am. Stat. Assoc. **44**, 335 (1949).
[45] E. George and R. McCulloch, J. Am. Stat. Assoc. **88**, 881 (1993).
[46] At this stage it may be convenient to introduce an approximation factor ϵ: Two circuits U and V are then considered to be equivalent if $||U - V|| \leqslant \epsilon$.
[47] http://www.tnt.uni-hannover.de/staff/rosenhahn/MCGS.zip.
[48] https://quantum-computing.ibm.com/composer.
[49] University of California, Irvine, Machine Learning Repository, D. Dua and C. Graff, http://archive.ics.uci.edu/ml.