# Choco-Q: Commute Hamiltonian-based QAOA for Constrained Binary Optimization

Debin Xiang† and Qifan Jiang† and Liqiang Lu* and Siwei Tan and Jianwei Yin*
Zhejiang University
{db.xiang, jiang7f, liqianglu, siweitan}@zju.edu.cn, zjuyjw@cs.zju.edu.cn

***Abstract*—Constrained binary optimization aims to find an optimal assignment to minimize or maximize the objective meanwhile satisfying the constraints, which is a representative NP problem in various domains, including transportation, scheduling, and economy. Quantum approximate optimization algorithms (QAOA) provide a promising methodology for solving this problem by exploiting the parallelism of quantum entanglement. However, existing QAOA approaches based on penalty-term or Hamiltonian simulation fail to thoroughly encode the constraints, leading to extremely low success rate and long searching latency.**

**This paper proposes Choco-Q, a formal and universal framework for constrained binary optimization problems, which comprehensively covers all constraints and exhibits high deployability for current quantum devices. The main innovation of Choco-Q is to embed the commute Hamiltonian as the driver Hamiltonian, resulting in a much more general encoding formulation that can deal with arbitrary linear constraints. Leveraging the arithmetic features of commute Hamiltonian, we propose three optimization techniques to squeeze the overall circuit complexity, including Hamiltonian serialization, equivalent decomposition, and variable elimination. The serialization mechanism transforms the original Hamiltonian into smaller ones. Our decomposition methods only take linear time complexity, achieving end-to-end acceleration. Experiments demonstrate that Choco-Q shows more than 235× algorithmic improvement in successfully finding the optimal solution, and achieves 4.69× end-to-end acceleration, compared to prior QAOA designs.**

## I. Introduction

The constrained binary optimization is to find an assignment of binary variables that minimize or maximize the objective function under the constraints. Typically, the constraints usually comprise multiple linear equations, namely equality, e.g., $C\vec{u} = \vec{c}$. There are various representative applications of this problem, such as facility location [17], project scheduling [25], portfolio optimization [6], political districting [16], and energy system optimization [38]. The constrained binary optimization has been demonstrated as an NP-hard problem that takes exponential time and space complexity for the classical solver to get the exact solution [21], [29].

Quantum computing stands as a revolutionary paradigm to handle this problem by taking advantage of quantum entanglement and superposition. Notably, the quantum approximate optimization algorithm (QAOA) is a class of variational algorithms, which consists of objective Hamiltonian and driver Hamiltonian to iteratively search the optimal solution [2]–[4], [18], [44], [45]. Mathematically, it encodes the objective function and constraints into Hamiltonians and updates the parameters during simulation. In particular, the penalty-based method is a natural idea that inserts the constraint into the objective function as penalty terms [5], [10], [44]. While, another approach is to encode the constraints into the driver Hamiltonian [22], [23], [47].

TABLE I
QAOA designs for constrained binary optimization.

| Constraint encoding | Penalty-based | | Driver-Hamiltonian-based | |
|---|---|---|---|---|
| **Methods** | A. Verma et al. [44] | Red-QAOA [45] | Yoshioka et al. [47] *cyclic Hamilt.* | Choco-Q *commute Hamilt.* |
| **Universality** | soft const. | soft const. | hard const. only part of linear | hard const. arbitrary linear |
| **In-constraints rate** | 0.03% | 0.07% | 0.67% | 100% |
| **Success rate** | 0.02% | 0.03% | 0.14% | 67.1% |
| **End-to-end latency** | 16.6s | 16.7s | 19.6s | 7.07s |

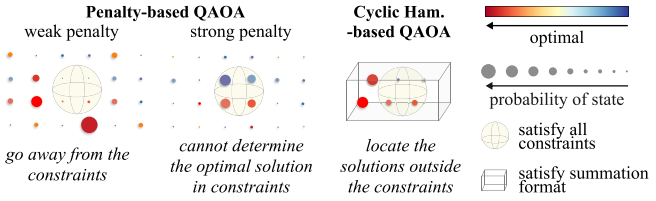
* The latency includes computing time and compilation time of QAOA, w/o data communication time. The execution time is estimated based on the IBM Fez model [15].

However, existing algorithms are severely limited by the low success rate, originating from the inability to thoroughly encode the constraints. Table I summarizes several features of previous QAOA designs and compares them on graph coloring problem [26] using a 15-qubit simulator. Here, we focus on two critical metrics: 1) *in-constraints rate*, the probability that the output solutions satisfy all constraints; 2) *success rate*, the probability of getting the optimal solution. We observe an extremely low success rate across all prior methods. Especially, the penalty-based methods even exhibit a close-to-zero success rate, smashing quantum advantages.

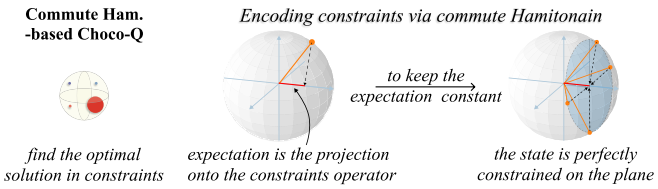
Fundamentally, the penalty-based method only supports soft constraint encoding, which highly depends on the coefficient of the penalty terms. Concretely, a small penalty coefficient may fail to take the constraints into account, as shown in Figure 1 (a). Nevertheless, a high coefficient will diminish the gap between the optimal solution and non-optimal ones when calculating the objective, decreasing the success rate. Thus, tuning the penalty coefficient gives rise to lots of time involved in classical computing. For example, in Table I, Red-QAOA [45] takes 16.7s, composed of the classical part and quantum part, and shows an inferior success rate of 0.03%.

Note that the penalty-based QAOA inherently expresses the Hamiltonian matrix described by the Ising model [27], which is a mathematical representation of the evolution of a

† These authors contributed equally to this work.
* Corresponding authors.




(a) The solution space under prior QAOA designs.




(b) The foundation of commute Hamiltonian-based approach.

Fig. 1. Comparison between Choco-Q and other QAOA designs.

quantum system. Therefore, an alternative approach to encode constraints is to develop other types of Hamiltonian, such as the quantum alternating operator ansatz that encodes the constraints into the driver Hamiltonian [22], [47]. However, prior Hamiltonian-based method has to restrict the constraints in a specific format, lacking the generality to address arbitrary linear equality. For example, the cyclic Hamiltonian-based simulation [47] only supports the equality described in summation format (e.g., $x_1+x_2+x_3=2$, or $-x_1-x_2-x_3=-1$). Thus, when coping with complex constraints, it may locate solutions in the non-constrained space, as shown in Figure 1 (a). These approaches also exhibit poor success rates, as shown in Table I. Moreover, the implementation of Hamiltonian requires decomposing it into deployable basic gates, accounting for exponential time complexity, leading to long end-to-end latency.

In this work, we propose Choco-Q, a formal and universal framework that smoothly (like a chocolate) tackles all these limitations. The key novelty of it lies on an expressive representation for encoding the constraints — commute Hamiltonian. In quantum mechanics, Heisenberg's picture states that if the Hamiltonian commutes with an operator, the expectation of the operator remains invariant during the evolution [24]. As illustrated in Figure 1 (b), the expectation value of the operator is calculated by projecting the quantum state onto the operator axis. To maintain a constant expectation value, the quantum states are constrained on the plane orthogonal to the operator axis. With this understanding, we can deduce that if a Hamiltonian commutes with the constraints operator, the states will be restricted to a subspace where each state adheres to the constraints. Starting with the commute Hamiltonian, we comprehensively reformulate the QAOA algorithm flow, which is capable of precisely covering arbitrary linear constraints. Furthermore, restricting the evolution under the commute operator substantially shrinks the search space, bringing a great reduction in the number of iterations.

Then, we present three optimization passes to facilitate end-to-end acceleration and make Choco-Q deployable on current NISQ devices. First, we propose to decouple the commute Hamiltonian into a series of local Hamiltonian, preventing massive tensor computation in the classical part. Unlike the prior Hamiltonian which has to repeatedly execute the Hamiltonian for approximation, we only need to execute one block with much less circuit depth. Though we serialize the entire Hamiltonian into smaller ones, it still has $4^N$ degrees of freedom, leaving a huge unitary decomposition overhead for generating high-quality circuits. To handle this, we design a novel unitary decomposition for the commute Hamiltonian, which achieves linear complexity in both time and circuit depth while keeping strict equivalence. Finally, we employ a variable elimination technique to further reduce the circuit depth from thousands to hundreds.

The main contributions of this paper include:

- We propose a formal, general, and deployable quantum algorithm for constrained binary optimization problems, which leverages the commute Hamiltonian to encode the constraints. This is the first work that accomplishes the goal of a fast and high-confident solver for this problem.
- We propose a series of optimization techniques to improve the deployability of the commute Hamiltonian on quantum hardware, including Hamiltonian serialization, equivalent decomposition, and variable elimination. We demonstrate the effectiveness and reliability of these optimizations by introducing two lemmas, which theoretically ensure the linear complexity in both decomposition time and circuit depth.
- We conduct rigorous experiments on real-world quantum devices using three practical application scenarios: facility location, graph coloring, and K-partition problem. Choco-Q shows remarkable success rates and achieves end-to-end speedup compared to prior works.

In algorithmic level, for the problems that prior methods can find the optimal solution, Choco-Q increases the success rate by $235\times$ compared to the state-of-the-art QAOA algorithm that applies cyclic Hamiltonian [47]. For the problems that prior methods fail to solve, Choco-Q still exhibits 9.5% - 54% success rates. On real-world quantum hardware, Choco-Q improves the success rate by $2.65\times$ and achieves a $4.69\times$ speedup compared to [47]. Besides, we reduce the decomposition time over $10^6\times$ and reduce the circuit depth more than $10^4\times$, compared to the existing compilation methods [1], [33], [36]. Choco-Q is publicly available on https://github.com/JanusQ/Choco-Q.git.

## II. Background

### A. Constrained Binary Optimization

The constrained binary optimization problem aims to find an optimal assignment of binary variables to minimize or maximize the objective function meanwhile satisfying several constraints. The objective function takes the binary variables

as inputs and returns a scalar. The constraints are represented using linear equations,

$$\begin{aligned} \min_{\vec{x}} \text{ or } \max_{\vec{x}} \quad & f(\vec{x}), \quad \vec{x} = \{x_1, x_2, \cdots, x_n\} \\ s.t. \quad & C\vec{x} = \vec{c}, \quad \vec{x} \in \{0,1\}^{\otimes n} \end{aligned} \tag{1}$$

where $C$ is the constraint matrix that contains the coefficients of each linear equation. Figure 2 (a) gives an example with four binary variables and two constraint equations. And the optimal solution to maximize the objective function is $\{1, 0, 1, 0\}$. The time complexity of classical algorithms grows exponentially with the number of variables and constraints, e.g., the worst-case time complexity is $O(2^n)$ using integer linear programming [30].

### B. QAOA Preliminaries

For binary optimization, the QAOA algorithm puts each variable in a qubit and prepares a parameterized quantum circuit to encode the objective function. Generally, it consists of four steps, as shown in Figure 2 (b). **Step 1** initializes the input states through a layer of H gates, which generates a uniform distribution across all possible variable assignments. **Step 2** constructs the Hamiltonian $H^o$ of the objective function by substituting the variable $x_j$ with $\frac{I_j - Z_j}{2}$ in the objective, where $I_j$ and $Z_j$ are the identity operator and the Pauli Z operator on qubit $j$, respectively. The driver Hamiltonian comprises a set of $R_X$ gates, which is used to parameterize the variable assignments for variational optimization. **Step 3** simulates Hamiltonian by executing the quantum circuit after compilation. This circuit involves multiple layers of unitary $e^{-i\gamma_l H^o}$ and $R_X(\beta_l)$ gates, where $\gamma_l$ and $\beta_l$ are the parameters for the $l$-th layer. **Step 4** measures all qubits and generates a sequence of bitstrings representing the variable assignments. Specifically, the QAOA algorithm calculates the results of the objective function under these assignments. Then, it iteratively updates the parameters in Hamiltonian simulation to approximate the optimal solution, e.g., using gradient descent [8] or linear approximation method [39].

A natural idea to support constrained binary optimization is to insert the constraints into the objective function as a penalty term [44] (we call it *soft constraint*). For example, in Figure 2 (c), the constraints are formulated by introducing a positive coefficient $\lambda$.

$$\lambda[(x_1 - x_3)^2 + (x_1 + x_2 + x_4 - 1)^2]$$

Theoretically, to ensure the constraints are satisfied, this penalty term should be zero when converging to the optimal solution. During Hamiltonian simulation, an additional penalty Hamiltonian $H^p$ is required, which can be calculated by substituting $x_j = \frac{I_j - Z_j}{2}$ into the penalty term.

An alternative approach is to design a new driver Hamiltonian for the constraints (we call it *hard constraint*). For example, the cyclic Hamiltonian [47] is able to deal with the summation format of the constraints, which is inspired by the

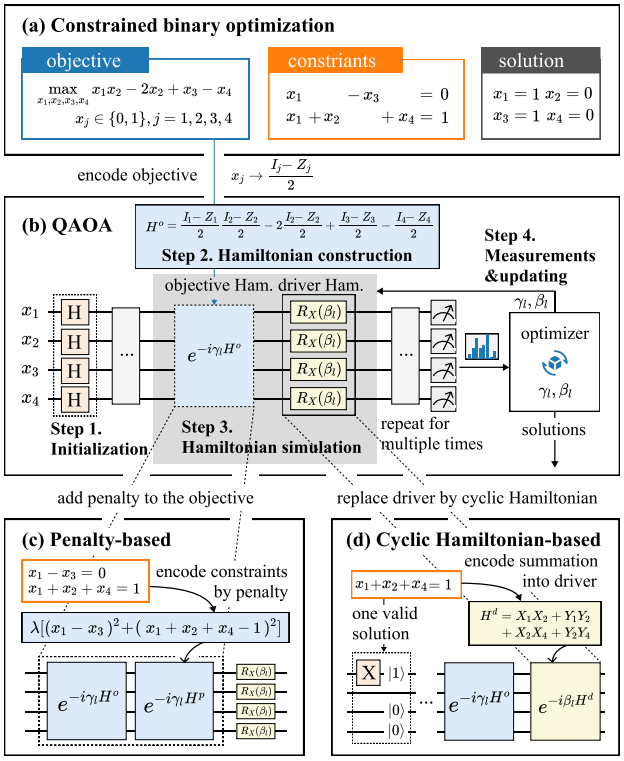



Fig. 2. Solving a constrained binary optimization using QAOA-based designs. (a) The example of a constrained binary optimization problem. (b) A typical QAOA flow that encodes objective into objective Hamiltonian, without the support of encoding constraints. (c) Extending QAOA by adding penalty terms to solve the constrained problem. (d) Applying cyclic Hamiltonian to encode constraints (only in summation format) into driver Hamiltonian $H^d$.

one-dimensional Ising model [27].

$$H^d = \sum_{i=1}^{n-1} X_i X_{(i+1)} + Y_i Y_{(i+1)} \tag{2}$$

Here $X_i$ and $Y_i$ are the Pauli-X and Pauli-Y operators on qubit $q_i$, respectively, which are determined by the variables in constraint. For example, the constraint $x_1 + x_2 + x_4 = 1$ involves variables $x_1, x_2, x_4$, and its cyclic Hamiltonian is

$$H^d = X_1X_2 + Y_1Y_2 + X_2X_4 + Y_2Y_4$$

In addition, we need to set the initial state to the solution of the constraint equation. For example, one solution of the constraint equation $x_1 + x_2 + x_4 = 1$ is $x_1 = 1, x_2 = 0, x_4 = 0$, thereby setting the three qubits to $|100\rangle$ as shown in Figure 2 (d).

## III. Choco-Q Formulation

Compared to penalty-based methods, encoding to driver Hamiltonian allows hard constraints and higher accuracy. However, the cyclic Hamiltonian is fundamentally limited by the summation format of the constraints that all coefficients must have the same sign, e.g., $x_1 + x_3 = 1$, $-x_1 - x_2 = -1$, and different constraint equations cannot share the same variables. Such limitation arises from the fact that the cyclic Hamiltonian, derived from the Ising model, has to keep the number of excited states $|1\rangle$ constant to encode the constraint.

Inherently, QAOA is a discrete simulation of the quantum evolution under objective Hamiltonian and driver Hamiltonian. Furthermore, for a linear constraint $\sum_{i=1}^{n} c_i x_i = c$, its operator is defined as,

$$\hat{C} = \sum_{i=1}^{n} c_i \sigma_i^z \tag{3}$$

where the driver Hamiltonian is responsible for maintaining the expected value of this operator constant during evolution. On the other hand, the Heisenberg picture states that if the driver Hamiltonian commutes with the operator, its expected value of the operator will remain unchanged [24]. With this understanding, we propose to employ the commute operator as driver Hamiltonian, namely commute Hamiltonian, which turns out to be a more general QAOA algorithm that enables arbitrary linear constraints for binary optimization, facilitating accuracy, scalability, and deployability.

### A. Applying Commute Hamiltonian to QAOA

The commute Hamiltonian supports hard constraints that can always satisfy the constraints. Commute Hamiltonian originates from the Heisenberg picture [24], which is an equal expression of the Schrödinger equation [14]. The Heisenberg picture gives that for any operator $\hat{A}$, the variation rate $\frac{d\hat{A}}{dt}$ under the evolution of Hamiltonian $H$ is proportional to the commutation operator between $\hat{A}$ and $H$.

$$\begin{aligned} \frac{d\hat{A}}{dt} &= \frac{i}{\hbar}[\hat{A}, H] \\ [\hat{A}, H] &= \hat{A}H - H\hat{A} \end{aligned} \tag{4}$$

Here, $[\hat{A}, H]$ is the commutation operator. $[\hat{A}, H] = 0$ leads to $\frac{d\hat{A}}{dt} = 0$, which implies that the expected value of $\hat{A}$ will not change during the evolution of $H$. In other words, we can find a driver Hamiltonian $H^d$ that commutes with the constraint operator $\hat{C}$ for constrained binary optimization $[\hat{C}, H^d] = 0$. In QAOA, the Hamiltonian $H$ comprises the objective Hamiltonian $H^o$ and the driver Hamiltonian $H^d$, denoted as $H = H^o + H^d$. Note that arbitrary objective Hamiltonian $H^o$ commutes with $\hat{C}$. This is because $H^o$ only involves $I$ and $\sigma^z$ operators, which always commute with the $\sigma^z$ operator in $\hat{C}$. Next, our goal is to find the driver Hamiltonian $H^d$ that also commutes with $\hat{C}$. Hannes et al. [32] gives a universal equation to find the commute Hamiltonian for specific constraints.

$$\begin{aligned} &H^d = \sum_{\vec{u}\in\Delta} H_c(\vec{u}) = \sum_{\vec{u}\in\Delta} (\sigma_1^{u_1} \cdots \sigma_n^{u_n} + \sigma_1^{-u_1} \cdots \sigma_n^{-u_n}) \\ &C\vec{u} = \vec{0},\ \vec{u} = \{u_1, u_2, u_3, ..., u_n\},\ u_i \in \{1, 0, -1\} \\ &\sigma_i^{+1} = \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix},\ \sigma_i^{0} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix},\ \sigma_i^{-1} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \end{aligned} \tag{5}$$

where $\Delta$ is the set of all valid solutions of $\vec{u}$ that satisfies $C\vec{u} = 0$, and $H_c(\vec{u})$ is the commute Hamiltonian for each valid $\vec{u}$.

Figure 3 depicts the basic flow of applying commute Hamiltonian to QAOA. Similarly, the initial quantum states are set to one solution of the constraint equation. The main

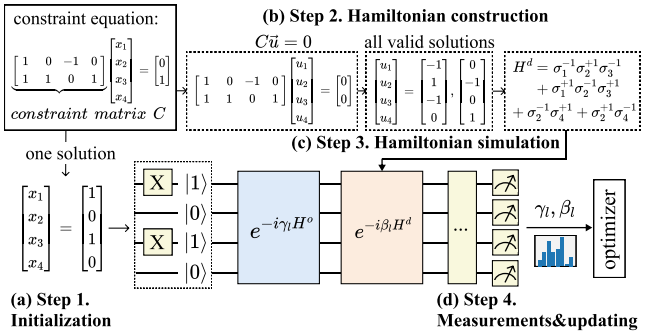



Fig. 3. The workflow of encoding the constraints via commute Hamiltonian.

difference lies in the design of the commute Hamiltonian and its simulation. Since the constraints are encoded by setting the commutation operator to zero $[\hat{C}, H^d] = 0$, all solutions of $C\vec{u} = 0$ form the commute Hamiltonian. For example, $\vec{u}^1 = [-1, 1, -1, 0]$ and $\vec{u}^2 = [0, -1, 0, 1]$ are all valid solutions for the case in Figure 3 (b), resulting in the following Hamiltonian according to Equation (5).

$$\begin{aligned} H^d &= H_c(\vec{u}^1) + H_c(\vec{u}^2) \\ H_c(\vec{u}^1) &= \sigma_1^{-1}\sigma_2^{+1}\sigma_3^{-1} + \sigma_1^{+1}\sigma_2^{-1}\sigma_3^{+1} \\ H_c(\vec{u}^2) &= \sigma_2^{-1}\sigma_4^{+1} + \sigma_2^{+1}\sigma_4^{-1} \end{aligned} \tag{6}$$

Consequently, the Hamiltonian simulation process is formulated by repeatedly executing the objective Hamiltonian $H^o$ and the commute Hamiltonian $H^d$.

$$\begin{aligned} |\phi_\theta\rangle &= \underbrace{e^{-i\beta_L H^d} e^{-i\gamma_L H^o} \cdots e^{-i\beta_1 H^d} e^{-i\gamma_1 H^o}}_{\text{repeat for } L \text{ times}} |\vec{x^*}\rangle \\ \theta &= \{\gamma_l, \beta_l\}_{l=1}^{L} = \{\beta_1, \gamma_1, \beta_2, \gamma_2, ..., \beta_L, \gamma_L\} \end{aligned} \tag{7}$$

Here, $|\vec{x^*}\rangle$ is the initial state in step 1 (e.g, it is $|1010\rangle$ Figure 3 (a)), $|\phi_\theta\rangle$ is the final state, and $\theta$ denotes $2L$ adjustable parameters for approximate optimization. From the perspective of commute Hamiltonian, the cyclic Hamiltonian can be regarded as a special case ($c_i \equiv 1$ in Equation (3)).

### B. Advantages and Challenges

Commute Hamiltonian is a much more general encoding scheme that deals with arbitrary linear constraints. By restricting the evolution under the commute operator, the search space significantly shrinks, resulting in a high-quality solution. More importantly, compared to the penalty-based QAOA, which often requires all-to-all entanglement between qubits, commute Hamiltonian exhibits sparse connectivity, making it hardware-friendly.

*Though the commute Hamiltonian shows higher generality and preciseness, it inevitably introduces computational overhead to obtain the accurate Hamiltonian.* According to Equation (5), the calculation of the commute Hamiltonian involves massive tensor-product and accumulation, further aggravating the overwhelming computational pressure. For $n$ qubits Hamiltonian, the calculation complexity is $O(2^n)$ and the time complexity is $O(n2^n)$.

*Like other QAOA approaches, the Hamiltonian unitary usually leads to huge circuit complexity after decomposition, making it challenging to deploy on NISQ devices.* Typically, existing decomposition methods show exponential complexity to approximate the unitary [12], [43], and often exhibit high approximation error [46]. For example, trotter decomposition [36] approximates the unitary $e^{-i\beta H^d}$ by dividing it into a series of small unitaries $e^{-i\beta H^d/N}$.

$$e^{-i\beta H^d} = \underbrace{e^{-i\beta H^d/N} \cdots e^{-i\beta H^d/N}}_{\text{repeat for N times}} \tag{8}$$

However, the accuracy of trotter decomposition depends on the number of $e^{-iH^d\beta/N}$ repetitions (the error is $O(1/N^2)$ after repeating N times). Moreover, these small unitaries still have to be decomposed into basic gates, severely increasing the circuit complexity and making it undeployable.

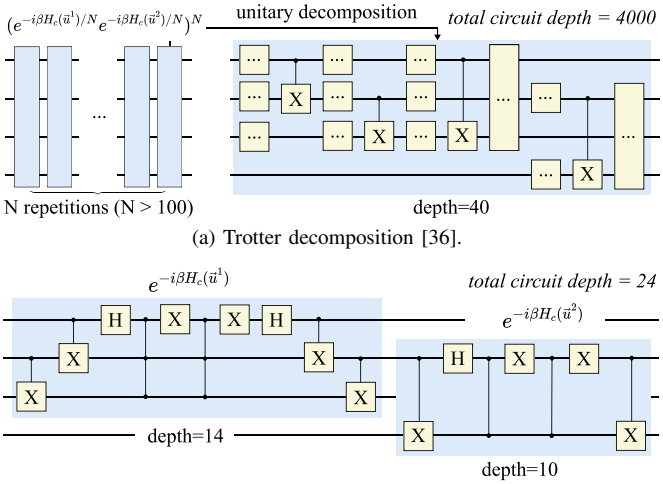
## IV. Choco-Q Optimization

To address the new challenges introduced by commute Hamiltonian, we propose multiple optimization passes. To deal with the overhead of tremendous classical computation when calculating the driver Hamiltonian, we serialize the commute Hamiltonian into a set of local Hamiltonian. Unlike the conventional simulation method that exhaustively partitions the Hamiltonian and repeatedly executes them, our serialization technique only needs to calculate the local Hamiltonian that takes less computational cost (Section IV-A). Each local commute Hamiltonian still requires to be decomposed into basic gates for deployment. Different from prior approximation-based unitary decompositions [1], [36], we present an equivalent transformation that decomposes the commute Hamiltonian into several control gates and phase gates. Using this, we demonstrate the linear complexity in both decomposition time and circuit depth (Section IV-B). By putting these two techniques together, we effectively prevent around GFlops tensor computation and reduce the circuit depth from $10^{10}$ to $\sim$1000. To further improve deployability on the current NISQ devices, we propose a variable elimination technique that can squeeze the constraint matrix, shrinking the circuit complexity (Section IV-C). Finally, we successfully achieve an executable circuit depth, around 100-depth, meanwhile exhibiting high accuracy for solving the constraint binary optimization problem.

### A. Serialization of Commute Hamiltonian

The commute Hamiltonian in Equation (5) exhibits highly-entangled complexity that requires connecting a large number of qubits, making it hard to deploy on sparsely-connected quantum chips. A natural idea is to reduce it into a series of smaller Hamiltonian, namely *local Hamiltonian* that acts only on a few qubits. However, note that when $H_1$ and $H_2$ are complex matrices, $e^{H_1+H_2} \neq e^{H_1}e^{H_2}$, which means that each item $e^{-i\beta H^d}$ in the driver Hamiltonian (Equation (7)) cannot be directly separated.

$$e^{-i\beta H^d} = e^{-i\beta\sum_{\vec{u}\in\Delta} H_c(\vec{u})} \neq \Pi_{\vec{u}\in\Delta} e^{-i\beta H_c(\vec{u})}$$




(a) Trotter decomposition [36].

(b) Serialization of commute Hamiltonian.

Fig. 4. Different Hamiltonian simulation methods for Equation (6).

Taking $\vec{u}^1 = [-1, 0], \vec{u}^2 = [-1, 1], \beta = 0.8$ as an example, we can easily verified that $e^{-i\beta(H_c(\vec{u}^1)+H_c(\vec{u}^2))} \neq e^{-i\beta H_c(\vec{u}^1)}e^{-i\beta H_c(\vec{u}^2)}$. Essentially, the function of the commute Hamiltonian lies on the encoding of the constraints. In other words, though the separation of Hamiltonian fails to keep the equivalence, we prove that it still meets the constraint operator of Equation (3), which provides the opportunity to shrink the circuit complexity.

**Lemma 1.** *Replacing the commute Hamiltonian unitary $e^{-i\beta H^d}$ with the serialization of unitaries $\Pi_{\vec{u}\in\Delta} e^{-i\beta H_c(\vec{u})}$ still satisfies the constraint operator during evolution.*

$$|x_c\rangle = e^{-i\beta H^d}|x\rangle, \ |x_s\rangle = \Pi_{\vec{u}\in\Delta} e^{-i\beta H_c(\vec{u})}|x\rangle$$
$$\langle x_s|\hat{C}|x_s\rangle = \langle x_c|\hat{C}|x_c\rangle$$

*Proof.* The expected value of the constraints operator $\hat{C}$ is calculated by the inner product $\langle x|\hat{C}|x\rangle$, where $|x\rangle$ is the initial state. The quantum state $|x_c\rangle$ after applying the commute Hamiltonian unitary satisfies the constraint during evolution:

$$\langle x_c|\hat{C}|x_c\rangle = \langle x|\hat{C}|x\rangle \tag{9}$$

Then, we use $\hat{B}$ to denote $\Pi_{\vec{u}\in\Delta} e^{-i\beta H_c(\vec{u})}$. The expected value of $\hat{C}$ after applying this serialized unitaries $\hat{B}$ can be calculated as follows,

$$\langle x_s|\hat{C}|x_s\rangle = \langle \hat{B}x| \ \hat{C}|\hat{B}x\rangle = \langle x|\hat{B}^\dagger \hat{C}\hat{B}|x\rangle \tag{10}$$

where $\hat{B}^\dagger$ denotes the conjugate transpose of $\hat{B}$, satisfying $\hat{B}^\dagger\hat{B}$ is the identity operator. On the other hand, we can easily verify that each Hamiltonian $H_c(\vec{u})$ commutes with $\hat{C}$.

$$[H_c(\vec{u}), \hat{C}] = 0$$

Furthermore, since the Hamiltonian unitary can be expanded by its Taylor series,

$$e^{-i\beta H_c(\vec{u})} = \textstyle\sum_{k=1}^{\infty}(-i\beta H_c(\vec{u})/k!)^k$$

and we have proven that each term $H_c(\vec{u})$ all commute with $\hat{C}$,

the Hamiltonian unitaries then commute with $\hat{C}$. Thus, their product also commutes with $\hat{C}$, which implies $\hat{C}\hat{B} = \hat{B}\hat{C}$. Substituting it into Equation (10), we have

$$\langle x_s| \hat{C} |x_s\rangle = \langle x| \hat{B}^\dagger \hat{C} \hat{B} |x\rangle = \langle x| \hat{B}^\dagger \hat{B} \hat{C} |x\rangle = \langle x_c| \hat{C} |x_c\rangle \quad \square$$

Lemma 1 gives the theoretical foundation for the serialization of commute Hamiltonian, which is specific to the constraint binary optimization problem. Compared to conventional Hamiltonian simulation methods [35], our approach significantly reduces the circuit depth from thousands to dozens, making it possible to implement the circuit. For example, Figure 4 (a) shows trotter decomposition [36] that is widely used for implementing the driver Hamiltonian (Equation (8)). This method cuts the Hamiltonian into N pieces and repeatedly executes them, leading to tremendous gates and huge circuit depth. In contrast, Figure 4 (b) depicts the circuit that embeds the constraints using the serialization of commute Hamiltonian. We effectively eliminate the repetition of massive small Hamiltonian, reducing the circuit depth from 4000 to 24 (after decomposition). Moreover, our method exhibits less connection between qubits, which is more hardware-friendly when compiling the circuit for a certain topology. For instance, $H_c(\vec{u}^1)$ acts on three qubits $q_1, q_2, q_3$, and $H_c(\vec{u}^2)$ only involves two qubits $q_2, q_4$.

### B. Decomposition of Commute Hamiltonian

Though we serialize the large commute Hamiltonian into a set of smaller Hamiltonian, it still requires decomposing these unitaries into basic gates. In particular, the decomposition quality highly determines the final accuracy and the deployability, necessitating a precise and fast methodology. To achieve this, we identify that applying the local commute Hamiltonian to its eigenstate fundamentally behaves like a phase gate,

$$\begin{aligned} e^{-i\beta H_c(\vec{u})} |x^\pm\rangle &= e^{\mp i\beta} |x^\pm\rangle \\ e^{-i\beta H_c(\vec{u})} |other\rangle &= |other\rangle \end{aligned} \tag{11}$$

where $|x^\pm\rangle$ are the eigenstates.

$$\begin{aligned} |x^\pm\rangle &= \frac{|v_1 \cdots v_n\rangle \pm |\bar{v}_1 \cdots \bar{v}_n\rangle}{\sqrt{2}} \\ v_i = \frac{1+u_i}{2}, \bar{v}_i &= \frac{1-u_i}{2} \end{aligned} \tag{12}$$

Inspired by this property, we first introduce an equivalent transformation that represents the Hamiltonian into multi-phase control gates. Then, we demonstrate the linear complexity in both time and circuit depth.

**Lemma 2.** *Each commute Hamiltonian $e^{-i\beta H_c(\vec{u})}$ can be equivalently decomposed as follows.*

$$e^{-i\beta H_c(\vec{u})} = G^\dagger P(\beta) X_1 P(-\beta) X_1 G \tag{13}$$

*Here, $X_1$ gate refers to an X gate applied on the first qubit $q_1$, $G$ is a set of gates to convert the eigenstate $|x^+\rangle$ and $|x^-\rangle$ into basis state $|01\cdots1\rangle$ and $|11\cdots1\rangle$, respectively.*

$$G|x^+\rangle = |01\cdots1\rangle, G|x^-\rangle = |11\cdots1\rangle \tag{14}$$

**Algorithm 1:** Implementation of converting gates $G$

**Input:** solution $\vec{u} = \{u_1, u_2, \cdots, u_n\}$ of $C\vec{u} = 0$.
**Output:** the circuit to implement $G$ gates in Equation (14).

```
1: for i = 1 to n do
2:     v_i = (1 + u_i)/2. // Equation 12
3: end for
4: // turn last n-1 qubits into |11...1⟩
5: for i = n to 2 do
6:     CX with target qubit q_i and control qubit q_{i-1}.
7:     if v_i = v_{i-1} then
8:         applying X on qubit q_i.
9:     end if
10: end for
11: // current state is |s^±⟩ = (|0⟩ ± |1⟩)|1...1⟩
12: applying H on the first qubit.// |s^{+(-)}⟩ → |0(1)1...1⟩
```

*$P(\beta)$ is a multi-control phase gate that only adds a phase $e^{i\beta}$ to the state $|1\cdots1\rangle$ and keeps the other eigenstates as same.*

$$\begin{aligned} P(\beta)|1\cdots1\rangle &= e^{i\beta}|1\cdots1\rangle \\ P(\beta)|other\rangle &= |other\rangle \end{aligned} \tag{15}$$

*Proof.* We prove this lemma by demonstrating that the decomposition in Equation (13) maintains the same eigenstates $|x^\pm\rangle$. According to Equation (14), taking $|x^+\rangle$ as an example,

$$\begin{aligned} &G^\dagger P(\beta) X_1 P(-\beta) X_1 G |x^+\rangle \\ =&G^\dagger P(\beta) X_1 P(-\beta) X_1 |01\cdots1\rangle \end{aligned}$$

applying $X_1$ gate on qubit $q_1$ will change $|01\cdots1\rangle$ to $|11\cdots1\rangle$. According to Equation (15), the phase gate $P(-\beta)$ will put a phase $e^{-i\beta}$ on this state.

$$P(-\beta)X_1|01\cdots1\rangle = e^{-i\beta}|11\cdots1\rangle$$

Similarly, according to the definition of $G^\dagger$ and $P(\beta)$, we can verify that,

$$G^\dagger P(\beta) X_1 P(-\beta) X_1 G |x^+\rangle = e^{-i\beta}|x^+\rangle$$

Furthermore, since the other states cannot activate the multi-control phase gate this decomposition will keep the other eigenstates unchanged.

$$G^\dagger P(\beta) X_1 P(-\beta) X_1 G |other\rangle = |other\rangle \quad \square$$

After decomposing the commute Hamiltonian into several convert gates $G$ and phase gates $P(\beta)$, we then illustrate that compiling these gates to basic gates (e.g., Hadamard, $R_Z$, CX) only takes linear time complexity and linear circuit depth. The implementation of the convert gate $G$ is shown in Algorithm 1. First, the eigenstate $|x^\pm\rangle$ is calculated using $v_i$ in Equation 12 (line 1-3). The core idea is to transform the last $n-1$ qubits to $|1\rangle$ state (line 4), which is implemented by CX and X gates (lines 5-10). Clearly, the CX gate helps to flip the target qubit if the control qubit is $|1\rangle$, while the X gate flips the state when two qubits are in the same state. After this, the quantum state is manipulated to a reduced state $|s^\pm\rangle$.

$$|s^\pm\rangle = (|0\rangle \pm |1\rangle)|1\cdots1\rangle/\sqrt{2} \tag{16}$$

Finally, we use an H gate on the first qubits to get $|01\cdots1\rangle$ for

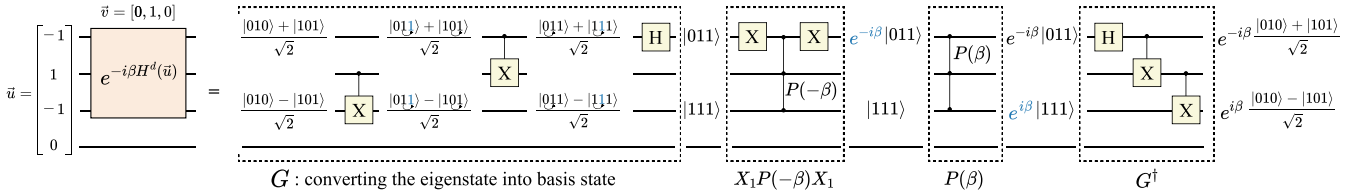
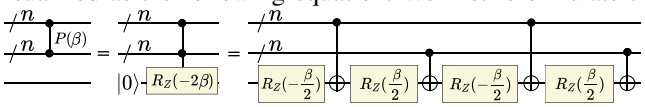



Fig. 5. The end-to-end decomposition flow of the commute Hamiltonian $e^{-i\beta H_c(\vec{u}^1)}$, following the same example in Equation 6.

$|x^+\rangle$ and $|11\cdots1\rangle$ for $|x^-\rangle$, respectively. Overall, according to Algorithm 1, the time complexity and circuit depth are with $O(n)$ complexity that is linear to the number of qubits in commute Hamiltonian.

The decomposition of the multi-quit phase gate can be visualized as the following equation. We first reformulate the




phase gate $P(\beta)$ as an $R_Z$ gate controlled by $2n$ qubits with an ancillary qubit in $|0\rangle$. Next, the control-$R_Z$ gate, with $2n+1$ qubits, can be further implemented by four control-X gates and four $R_Z$ gates, with each control-X gate involving $n+1$ qubits. Since the control-X gates can be implemented with $16n$-qubit gate with one ancillary qubit [19], the overall complexity of decomposing the $n$-qubit phase gate is $O(n)$. On the other hand, there are two ancillary qubits in total, which can be reused in the entire circuit of commute Hamiltonian simulation. Thus, the overall circuit complexity is also $O(n)$ with only two ancillary qubits.

Unlike prior approximation-based unitary decomposition [1], [9], [43], we design a precise decomposition formulation for implementing the commute Hamiltonian with linear complexity. Instead of numerically calculating the Hamiltonian through tensor computations in Equation (5), we can directly derive the decomposed circuit from the solution vector $\vec{u}$. Following the example $H_c(\vec{u}^1)$ in Equation (6), we can calculate the eigenstate according to Equation (12).

$$\vec{u}^1 = [-1, 1, -1, 0]$$
$$\text{Eq (12)} \Rightarrow \quad |x^\pm\rangle = (|010\rangle \pm |101\rangle)/\sqrt{2}$$

Figure 5 provides the end-to-end decomposition flow for this Hamiltonian. The $G$ gates are constructed by two CX gates and an H gate. Since $q_2$ and $q_3$ are not in the same state, the first CX gate sets the qubit $q_3$ into $|1\rangle$. Similarly, the second CX gate turns qubit $q_2$ into $|1\rangle$. Then, the H gate on qubit $q_1$ changes the state $(|0\rangle + |1\rangle)|11\rangle/\sqrt{2}$ to $|011\rangle$ and changes the state $(|0\rangle - |1\rangle)|11\rangle/\sqrt{2}$ to $|111\rangle$. The phase gate $X_1P(-\beta)X_1$ puts $e^{-i\beta}$ to state $|011\rangle$, and $P(\beta)$ puts phase $e^{i\beta}$ to state $|111\rangle$. After applying the inversion gate $G^\dagger$, we successfully add phase $e^{i\mp\beta}$ on state $|x^\pm\rangle$, constructing the whole circuit for the Hamiltonian $H_c(\vec{u}^1)$. Note that we also draw the resultant circuit of the overall driver Hamiltonian in Figure 4, illustrating the linear complexity of circuit depth.

### C. Variable Elimination for Higher Deployability

Though our decomposition method has greatly reduced the circuit depth, it is still intolerable for the current NISQ devices. For example, solving a facility location problem with 28 variables features a 989-depth circuit after Hamiltonian decomposition, which goes far beyond the executable depth ($\sim$100) in real-world quantum platforms. Besides, the limited number of qubits also aggravates the difficulty of deployment. An orthogonal optimization direction is to reduce the searching space, i.e., squeeze the constraint matrix by eliminating part of the variables.

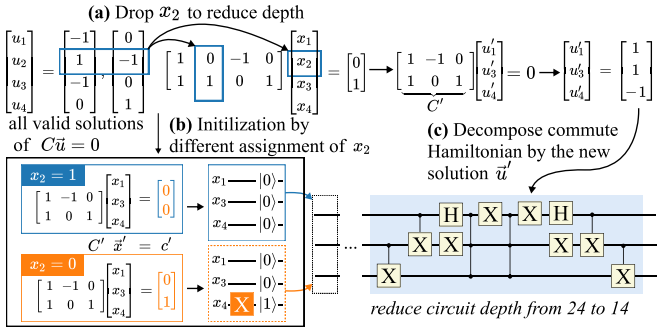
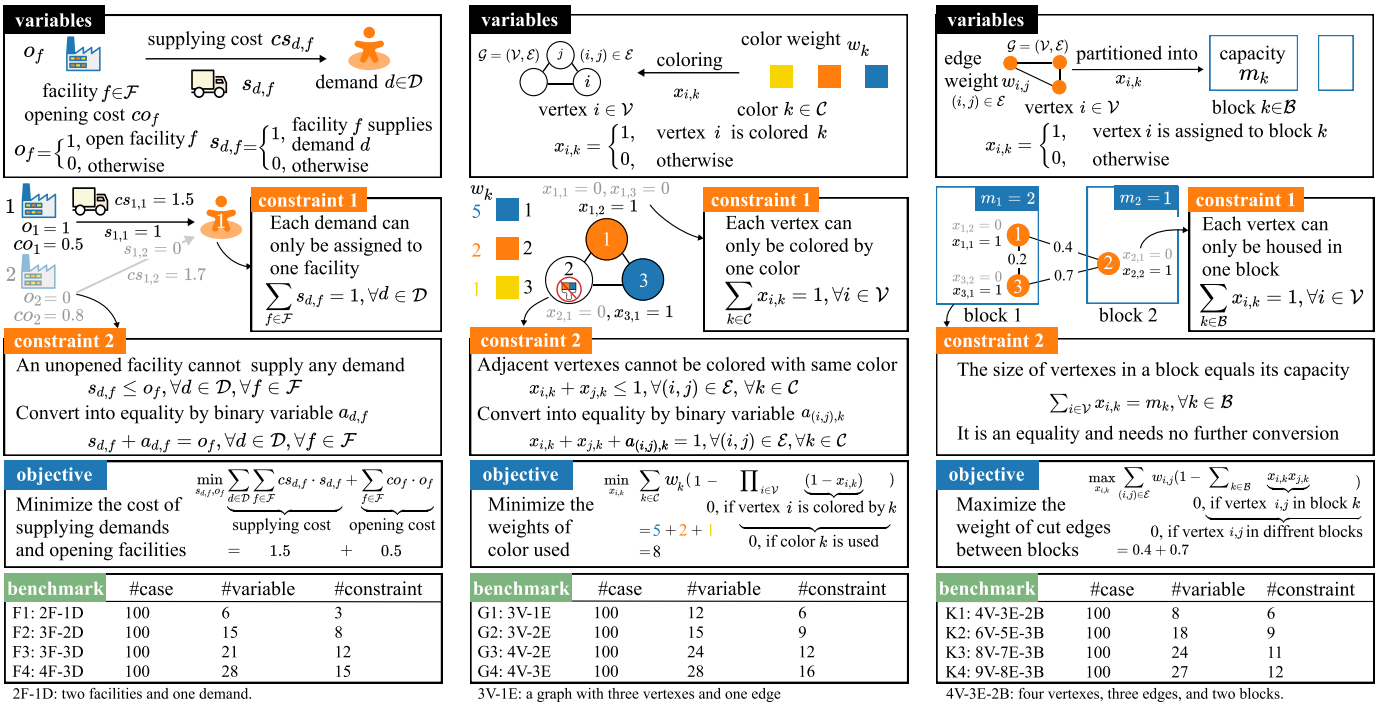



Fig. 6. Eliminating the variable that involves most non-zero elements across all solutions of $C\vec{u} = 0$, which brings higher deployability.

To this end, we further deeply investigate the relationship between circuit depth and the complexity of constraints. In Lemma 2, we have demonstrated that the circuit depth is linear to the number of qubits. Especially, the depth of each local Hamiltonian is linear to the non-zero elements in the solution $\vec{u}$. For example, in Figure 4 (b), $\vec{u}^1$ and $\vec{u}^2$ involves 3 and 2 non-zero elements, respectively. Thus, the circuit block of Hamiltonian $e^{-i\beta H_c(\vec{u}^1)}$ and $e^{-i\beta H_c(\vec{u}^2)}$ takes 3 and 2 qubits. *In conclusion, the circuit depth of commute Hamiltonian simulation is proportional to the total number of nonzero elements in all solution vectors $\vec{u} \in \Delta$ of $C\vec{u} = 0$.*

Based on this observation, we propose a variable elimination technique to reduce the number of variables, which further helps to reduce the dimensions of the solution vector $\vec{u}$. Figure 6 shows an example that eliminates the variable $x_2$, reducing the solution vector to a triple $\vec{u'} = \{u'_1, u'_3, u'_4\}$. And the driver Hamiltonian is then derived from the new constraint matrix $C'\vec{u'} = 0$. Accordingly, the number of non-zeros is reduced from 5 (3+2 in $\vec{u}^1$ and $\vec{u}^2$) to 3 ($\vec{u'}$). Consequently, the circuit depth and the required qubits are cut down, from 24-depth to 14-depth, and from 4 qubits to 3 qubits. However, such source saving introduces the overhead of more measurements, diminishing the quantum advantage. This approach requires enumerating the possible assignment of the eliminated variables and executing the circuit individually. For example, in Figure 6 (b), setting $x_2$ to 0 or 1 leads to




| benchmark | #case | #variable | #constraint |
|---|---|---|---|
| F1: 2F-1D | 100 | 6 | 3 |
| F2: 3F-2D | 100 | 15 | 8 |
| F3: 3F-3D | 100 | 21 | 12 |
| F4: 4F-3D | 100 | 28 | 15 |

2F-1D: two facilities and one demand.

| benchmark | #case | #variable | #constraint |
|---|---|---|---|
| G1: 3V-1E | 100 | 12 | 6 |
| G2: 3V-2E | 100 | 15 | 9 |
| G3: 4V-2E | 100 | 24 | 12 |
| G4: 4V-3E | 100 | 28 | 16 |

3V-1E: a graph with three vertexes and one edge

| benchmark | #case | #variable | #constraint |
|---|---|---|---|
| K1: 4V-3E-2B | 100 | 8 | 6 |
| K2: 6V-5E-3B | 100 | 18 | 9 |
| K3: 8V-7E-3B | 100 | 24 | 11 |
| K4: 9V-8E-3B | 100 | 27 | 12 |

4V-3E-2B: four vertexes, three edges, and two blocks.

(a) Facility location problem [37] (b) Graph coloring problem [26] (c) K-partition problem [11]

Fig. 7. The detailed description, formulation, and benchmark settings of three practical applications.

different initialization when solving $C'\vec{x}' = c'$. And we need to measure these two circuits to construct the final results.

Theoretically, the outputs after eliminating variables still satisfy the original constraints. The constraint $\sum_{i=1}^{n} c_i x_i = c$ is transformed into $\sum_{i \neq j} c_i x_i = c - c_j x_j$ after eliminating variable $x_j$. Consequently, the commute Hamiltonian is generated from the new constraint $\sum_{i \neq j} c_i u_i = 0$ with the initial state $x^o$ set by the solution of the new constraints $\sum_{i \neq j} c_i x_i^0 = c - c_j x_j$. This means $\sum_{i \neq j} c_i x_i' + c_j x_j = c$, which demonstrates that the results (including the evolved $x_j$) strictly satisfy the original constraints.

Actually, the measurement overhead increases exponentially to the number of eliminated variables. Such variable elimination is fundamentally an approach to make the circuit more deployable by sacrificing the theoretical parallelism of quantum computing. Therefore, we develop an identification method to find the variable that gives rise to a large reduction in the circuit depth. To be specific, we choose the variable that involves most non-zero elements across all solutions of $C\vec{u} = 0$. For example, in Figure 6 (a), in the solution of $\vec{u}^1$ and $\vec{u}^2$, $x_2$ features two non-zero assignments, making it an eliminable variable.

## V. Evaluation

### A. Experiment Setup

**Benchmark.** We evaluate Choco-Q using three practical application scenarios, including facility location problem (FLP) [37], graph coloring problem (GCP) [26], and k-partition problem [11]. Figure 7 provides detailed problem formulation with their objective functions and constraints. For each application, we collect 400 cases from the related literature [11], [26], [37]. And we categorize these cases into four problem scales (e.g., F1 to F4 in FLP), with the number of variables ranging from 6 to 28, and the number of constraints ranging from 3 to 16.

**Comparison.** We compare Choco-Q with previous QAOAs that support constrained binary optimization, including Penalty-based QAOA [44], cyclic Hamiltonian-based QAOA [47], and Hardware-efficient ansatz (HEA) [28]. For Penalty-based QAOA, We integrate it with two state-of-the-art QAOA optimization techniques, *FrozenQubits* [4] and *Red-QAOA* [45]. Specifically, *FrozenQubits* aims to reduce the circuit depth and enhance the success rate, while *Red-QAOA* optimizes initial parameters. HEA is a universal variational quantum algorithm (non-QAOA) [41], where we use the circuit provided by Kandala et.al [28]. When designing HEA, we introduce a penalty method to make the output satisfy the constraints as much as possible. For parameter updating, we use the constrained optimization by linear approximation method [39] for all designs.

**Platform.** We conduct several small-scale evaluations on three IBMQ systems, including *Fez* platform with 159-qubit Heron r2 type, *Sherbrooke* and *Osaka* platform with 127-qubit Eagle r3 type [15]. Since QAOA usually comprises a large amount of CZ gates, *Fez* is QAOA-friendly as it features the CZ gate as the basic gate with 99.7% fidelity. The other two devices only support single-direction ECR gates with 99.3% fidelity, which takes three ECR gates to implement the CZ gate, resulting in a higher error rate. The simulation experiments and the classical part of QAOA are executed on

TABLE II
THE CIRCUIT DEPTH, SUCCESS RATE, AND IN-CONSTRAINTS RATE OF DIFFERENT QAOA DESIGNS UNDER 12 BENCHMARKS.

| Benchmark | | Success rate (%) | | | | In-constraints rate (%) | | | | Approximation ratio gap (ARG) | | | | Circuit depth | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Penalty [44] | Cyclic [47] | HEA [28] | Choco -Q | Penalty [44] | Cyclic [47] | HEA [28] | Choco -Q | Penalty [44] | Cyclic [47] | HEA [28] | Choco -Q | Penalty [44] | Cyclic [47] | HEA [28] | Choco -Q |
| FLP [37] | F1 | 3.79 | 21.4 | 8.91 | **99.8** | 22.0 | 38.7 | 26.4 | **100.0** | 39.0 | 15.3 | 44.9 | **0.16** | 56 | 91 | 42 | 43 |
| | F2 | 0.14 | 0.09 | ✗ | **54.0** | 0.47 | 1.37 | 0.10 | **100.0** | 107 | 70.9 | 150 | **0.30** | 88 | 99 | 98 | 221 |
| | F3 | ✗ | 0.03 | ✗ | **30.0** | 0.04 | 0.79 | ✗ | **100.0** | 145 | 94.6 | 181 | **0.50** | 92 | 134 | 147 | 275 |
| | F4 | ✗ | ✗ | ✗ | **13.3** | ✗ | 0.74 | ✗ | **100.0** | 177 | 97.7 | 257 | **0.43** | 108 | 162 | 196 | 421 |
| GCP [26] | G1 | 0.15 | 4.74 | 0.26 | **69.7** | 0.50 | 10.6 | 0.36 | **100.0** | 698 | 217 | 1144 | **0.06** | 188 | 230 | 84 | 24 |
| | G2 | 0.03 | 0.14 | 0.03 | **67.1** | 0.07 | 0.67 | 0.03 | **100.0** | 747 | 621 | 1727 | **0.25** | 221 | 263 | 105 | 358 |
| | G3 | ✗ | ✗ | ✗ | **17.1** | 0.02 | 0.27 | ✗ | **100.0** | 2592 | 479 | 3091 | **0.47** | 654 | 708 | 168 | 586 |
| | G4 | ✗ | ✗ | ✗ | **9.50** | ✗ | ✗ | ✗ | **100.0** | 2366 | 501 | 3834 | **0.56** | 683 | 737 | 196 | 906 |
| KPP [11] | K1 | 1.66 | 38.2 | 1.04 | **86.1** | 5.18 | 84.8 | 4.46 | **100.0** | 53.8 | 32.3 | 59.4 | **0.14** | 115 | 150 | 56 | 79 |
| | K2 | 0.01 | 14.2 | ✗ | **52.6** | 0.05 | 39.7 | 0.04 | **100.0** | 118 | 37.8 | 130 | **0.18** | 202 | 244 | 126 | 324 |
| | K3 | 0.01 | 2.59 | ✗ | **21.1** | 0.01 | 31.6 | ✗ | **100.0** | 162 | 24.7 | 165 | **0.24** | 264 | 306 | 168 | 464 |
| | K4 | ✗ | 0.45 | ✗ | **13.3** | ✗ | 8.23 | ✗ | **100.0** | 163 | 30.5 | 170 | **0.23** | 296 | 338 | 189 | 534 |
| **Improv.(×)** | | - | - | - | **>235** | - | - | - | **>80.6** | - | - | - | **658** | - | - | - | **1.00** |

[1] For Penalty-based QAOA [44], we integrate it with two open-sourced optimization techniques, *FrozenQubits* [4] and *Red-QAOA* [45]. Choco-Q only eliminates one variable.
[2] The improvement is compared to the cyclic Hamiltonian-based method [47]. ✗ means that the design fails to find the optimal solution for this case.

an AMD EPYC 9554 64-core sever with 1.5T SSD memory. The simulation of the quantum circuit is accelerated by one A100 GPU on the server.

**Evaluation metrics.** Similar to prior constraint QAOAs, we employ three algorithmic metrics: *success rate*, *in-constraints rate*, and *approximation ratio gap*. The *success rate* is defined as the probability of getting the optimal solution after measurements. The *in-constraints rate* is the probability that the output solutions satisfy the constraints. Thus, the in-constraints rate is always higher than the success rate. *Approximation ratio gap* (ARG) indexes the solution quality across all outcomes generated by the algorithm, which is widely used in prior QAOAs [4], [20], [45]. It is defined as

$$ARG = |\frac{E(f(\vec{x}) + \lambda\|C\vec{x} - \vec{c}\|)}{f(\vec{x_{optimal}})} - 1| \quad (17)$$

Here, the objective function $f$ and constraints $C$ are defined in Equation (1). $\vec{x}$ is the outcome, $\vec{x_{optimal}}$ is the optimal solution, and $E$ means the expectation value on all outcomes. $\lambda$ is the penalty term, which is set to 10 here to balance the objective and constraints. When implementing on real-world devices, we further evaluate the end-to-end *latency* (without data communication), including the compilation time for Hamiltonian decomposition, circuit execution time, and the parameter updating time for iterative optimization.

### B. Algorithmic Evaluation

**Success rate.** Compared to other QAOA designs, Choco-Q achieves an average 235× improvement in successfully finding the optimal solution, as shown in Table II. In particular, for the problem in medium-scale (F2, G2, K2), all other approaches nearly fail to get the optimal solution, with a success rate below 15%. While, our approach exhibits a high success rate, from 52.6% to 67.1%. Even for large-scale problems, we still have a relatively high chance of finding the solution (9.50% - 30.0%). Such great improvement comes from the fact that the commute Hamiltonian effectively narrows down the search space, leading to a higher probability of getting the optimal solution.

**In-constraints rate.** When encoding the constraints, Choco-Q shows a 100% in-constraints rate, indicating that our algorithm comprehensively takes all constraints into account. This is also the reason that Choco-Q has a much higher success rate. The 100% in-constraints rate benefits from the generality of our encoding scheme that applies commute Hamiltonian to thoroughly represent the constraints. In contrast, the cyclic Hamiltonian restricts constraints to a summation format, making it challenging to consider all constraints, especially in large-scale problems. Besides, the cyclic Hamiltonian performs better on KPP benchmarks since the constraints of KPP are in summation format and involve fewer shared variables. The in-constraints rate of penalty-based QAOA [44] is highly determined by the number of constraints, where each constraint equation corresponds to one penalty term in the objective function. In contrast, Choco-Q is irrelevant to it as we consider the problem as the commute operator between the driver Hamiltonian and the constraints.

**Approximation ratio gap (ARG).** As shown in Table II, Choco-Q improves the ARG by 658×. Such improvement originates from that in Choco-Q, $\lambda|C\vec{x} - \vec{c}|$ in Equation (17) is constantly zero, while it can be huge using other methods. In particular, the ARG is larger in GCP benchmarks since GCP contains more complicated constraints. Even though, the ARG of Choco-Q is below 0.6.

**Circuit depth.** Choco-Q involves a little bit more circuit depth, notably in large-scale problems. First, as mentioned in Section IV-B, prior algorithms are fundamentally approximation-based approaches for optimization problems, which may easily fail to meet the constraints. However, Choco-Q lies on the precise encoding of arbitrary linear constraints, which requires implementing the commute Hamiltonian of all solution $\vec{u}$ in Equation 5, resulting in the linear complexity of circuit depth. Taking the G3 case as an example, it results in 12 $\vec{u}$ to precisely express the 12 constraint equations, giving rise to large circuit depth. The second reason is that we simulate the other QAOA algorithm only with seven repeated layers, i.e., repeat 7 times in Equation (7). We adopt seven layers because this setting achieves the best trade-off between success

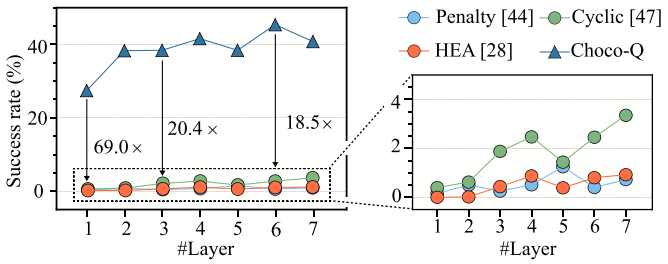



Fig. 8. The average success rate using the different number of layers.

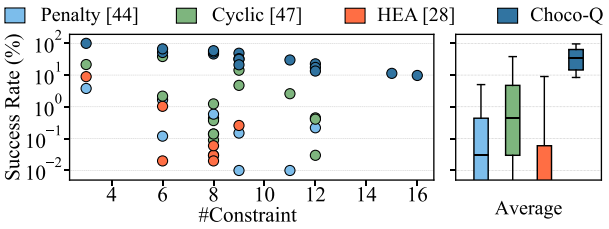
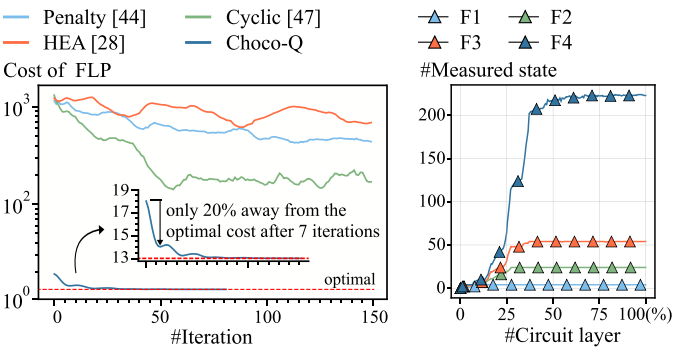



Fig. 9. The success rate and circuit depth of Choco-Q under the different number of constraints.




(a) Convergence using F1:2F-1D. (b) The parallelism of Choco-Q.

Fig. 10. Convergence analysis of Choco-Q.

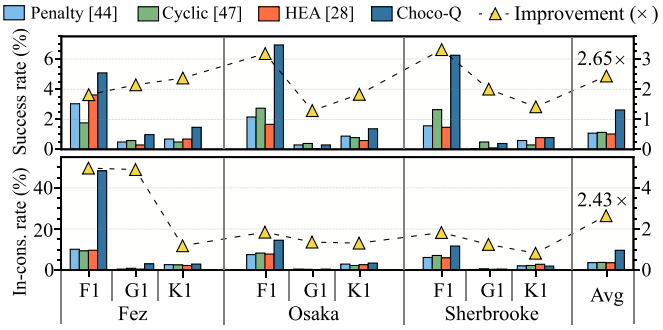



Fig. 11. Success rate and in-constraints rate on IBM quantum platforms.

rate and circuit depth. We test that even if we increase the number of layers, it only contributes to limited improvement in the success rate. And in Table II, we do not repeat the driver Hamiltonian of Choco-Q (one layer). In a word, compared to the significant performance improvement, the additional circuit depth is acceptable.

**Success rate with the different number of repeated layers.** Figure 8 evaluates the average success rate by varying the number of repeated layers. We can see that the success rate of our design is always over 25% while the others are less than 5%. In Choco-Q, applying two repeated commute Hamiltonian layers improves the success rate from 27.4% to 38.3%. Since the serialization of commute Hamiltonian has covered all searching directions, more layers can only bring limited improvement. On the contrary, increasing the number of layers significantly improves the success rate of other designs. For example, in the cyclic Hamiltonian-based QAOA [47], each additional layer leads to an average of 0.5% improvement. Even though, the final success rate is still hindered by the algorithmic limitations.

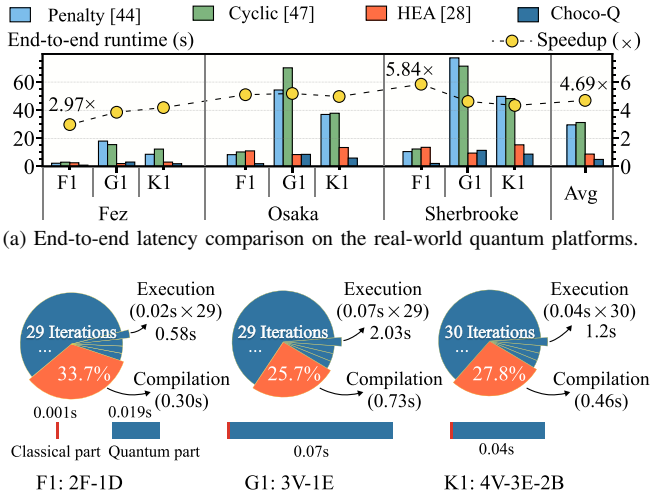
**Success rate with the different number of constraints.** Figure 9 plots the success rate of all graph benchmarks with the number of constraints on the x-axis. The advantage of Choco-Q becomes more significant with the growing number of constraints. In particular, when the number of constraints exceeds 12, the success rate of other methods almost becomes zero, while Choco-Q still holds a success rate of more than 10%. This achievement arises because the commute Hamiltonian strictly limits the search space under constraints.

**Convergence.** Figure 10 (a) depicts the convergence curves of different QAOA designs, picking one case from the F1:2F-1D benchmark. Choco-Q can reach the optimal cost within 30 iterations, while other methods require over 148 iterations and still have at least a 78% gap with the optimal cost. In particular, Choco-Q can quickly reach 20% away from the optimal cost within 7 iterations, which is 7.3× faster than the cyclic Hamiltonian-based approach. Such high convergence speedup is attributed to a good initial cost (18), while the others are extremely large ($10^3$). Furthermore, the squeezed search space also helps to find the optimal solution in fewer iterations. Figure 10 (b) illustrates the parallelism of Choco-Q. We collect the number of measured states through the circuit, which represents the parallelism involved in the superposition of quantum states. Choco-Q shows an exponential growth of the parallelism at the beginning of the circuit, i.e., at the point of around 1/4 circuit. Unlike prior QAOAs that initialize a uniform superposition state, even though Choco-Q prepares a special initial state, we still effectively harvest the quantum parallelism by applying commute Hamiltonian.

### *C. Evaluation on Real-world Quantum Platforms.*

This section evaluates Choco-Q on the IBM cloud quantum devices. Considering the short decoherence time and the inevitable noise error, we choose to implement the small-scale problems, i.e., F1, G1, and K1 cases in Table II.

**Success rate.** Figure 11 gives the success rate on three quantum devices. Compared to the noise-free simulator, the success rate of all cases is decreased due to various noise, e.g., cross-talk and hardware defects. Overall, Choco-Q achieves an average of 2.65× improvement over other designs. The effectiveness of NISQ devices arises from our decomposition technique that transforms the commute Hamiltonian into multiple phase gates, which are less sensitive to flip error. We observe that all methods on the G1 benchmark have the lowest success rate. This is because G1 requires 12 qubits to solve




(a) End-to-end latency comparison on the real-world quantum platforms.

(b) Latency breakdown of Choco-Q on the Fez platform.

Fig. 12. End-to-end latency evaluation and breakdown.

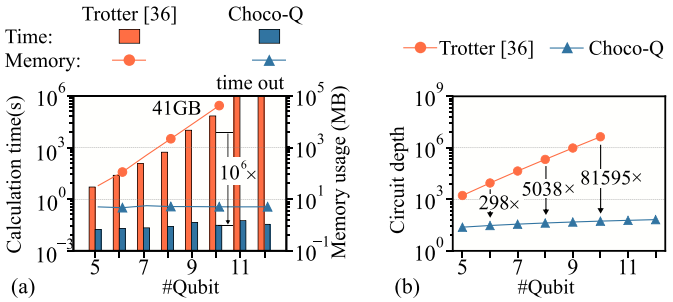
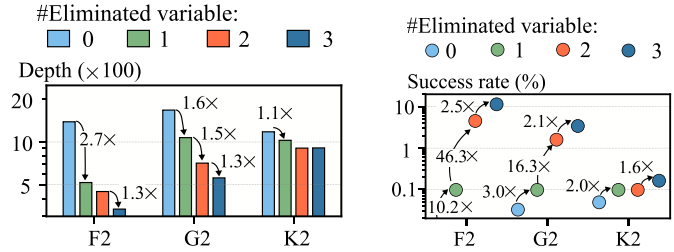



Fig. 13. Comparison between Trotter decomposition [36] and Choco-Q.




(a) Depth of circuit after eliminating the different number of variables.

(b) Success rate improvement under IBMQ noise model.

Fig. 14. Evaluation of variable elimination technique.

and involves more cross-talk errors. The circuits of F1 have a higher success rate since they only consist of six variables and three constraints.

**In-constraints rate.** Compared to other designs, Choco-Q shows 2.43× higher in-constraints rate. In particular, on the Fez platform, we achieve up to 48% in-constraints rate, resulting in 4.98× improvement, thanks to its architectural properties. As mentioned before, this device is QAOA-friendly, allowing us to fully leverage our technological advantages. As we serialize the driver Hamiltonian into a set of smaller ones that necessitates fewer qubit connections, Choco-Q is more noise-tolerant against the cross-talk noise.

**End-to-end latency.** Figure 12 (a) compares the overall latency comprising of the compilation time, circuit execution time, and parameter updating time. Choco-Q achieves 2.97× - 5.84× speedup, and always within 10 seconds, primarily attributed to the reduced number of iterations. The latency of HEA [28], the non-QAOA algorithm, is close to ours in several cases. This is because it shows shallow circuit depth in F1, G1, and K1 cases, leading to around 4× speedup when executing the circuit.

Figure 12 (b) presents the latency breakdown for the cases in the Fez platform. We consider the end-to-end latency as the summation of compilation time (including the decomposition) and execution time. The execution time includes the classical and quantum parts of QAOA, with the classical part only taking a little time. We can see that the iterative execution is the most time-consuming, requiring around 30 iterations and accounting for around 70% of the total latency.

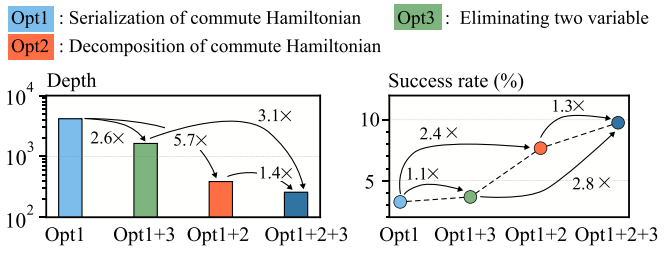
### *D. Detailed Analysis of Optimization Techniques*

**Decomposition of commute Hamiltonian.** Figure 13 compares the Hamiltonian decomposition results between Trotter decomposition [36] and Choco-Q. Figure 13 (a) shows the decomposition time and memory usage across different circuit sizes. By putting our serialization technique and decomposition formulation together, our approach demonstrates a linear time complexity (less than 0.1s) and constant memory usage (less than 10MB). Taking the 10-qubit Hamiltonian as an example, we achieve $10^6\times$, $8341\times$ reduction for the decomposition time and memory usage, respectively. When the number of qubits is beyond 10, the Trotter decomposition fails to output the results. The high scalability mainly stems from that we effectively eliminate the massive tensor computation in Equation (5) by serialization and equivalent transformation.

Figure 13 (b) compares the resultant circuit depth using these two decomposition methods. The circuit depth of Choco-Q is linearly increased with the number of qubits, from 24 (5-qubit) to 66 (12-qubit). Such quantum resource saving benefits from our serialization of commute Hamiltonian, which avoids enormous repetition of approximation Hamiltonian. In addition, our decomposition flow in Figure 5 also ensures that the convert gate and the phase gate can be efficiently decomposed into basic gates.

**Variable elimination.** We analyze the circuit depth and the success rate after eliminating a different number of variables, as shown in Figure 14. Overall, variable elimination produces a remarkable boost in both circuit depth and success rate. For example, in F2 case, just eliminating one variable leads to a 2.7× circuit depth reduction and yields 10.2× success rate improvement, which exceeds the overhead caused by extra measurements. We also find that such yields slow down when more eliminated variables are introduced. This is because most of the non-zero values in solutions of $C\vec{u} = 0$ have been eliminated and further elimination leads to less benefit. Besides, this technique has little gain in KPP cases due to the uniform distribution of qubits in the local Hamiltonian. Thus, as shown in Figure 6, dropping one of them cannot greatly reduce the number of non-zeros in all solutions of $C\vec{u} = 0$.

Moreover, to demonstrate the advantage of variable elimina-




(a) Ablation study of circuit depth. (b) Ablation study of success rate.

Fig. 15. Ablation study under IBMQ noise model.

tion, we investigate the success rate under the noise model of the three IBM quantum platforms. We can see that the success rate benefits most when eliminating the first two variables. For example, in F2, the success rate is increased by $10.2\times$ and $46.3\times$, respectively, leading to the success rate improved from 0.1% to 4.6%. Interestingly, the elimination of the 3rd variable generates limited improvement, from $1.6\times$ to $2.5\times$, which is mainly because most of the non-zero values have been eliminated. Even though this introduces $2\times$ overhead of more measurements, it is acceptable, especially considering the deployability on current NISQ devices.

### E. Ablation Experiments

Figure 15 gives the circuit depth and success rate under different configurations of our proposed optimization techniques. The metrics including depth and success rate are average over the benchmarks and the quantum devices. We have illustrated in Section V-D that the circuit without serialization of commute Hamiltonian (Opt1) cannot be executed on real-world devices, so we use opt1 in each configuration. Figure 15 (a) shows that eliminating one variable (Opt3) can reduce circuit depth by $2.6\times$(Opt1+3). Decomposing the local Hamiltonian by phase gate (Opt2) can reduce the circuit depth by $5.7\times$ (Opt1+2) compared to directly decomposing the local Hamiltonian unitary. The circuit depth can be further reduced by $1.4\times$ using variable elimination. The success rate improvement shows a similar trend with the circuit depth as shown in Figure 15 (b). Decomposition of commute Hamiltonian can improve success rate by $2.4\times$ (Opt1+2). Variable elimination can further increase success rate by $1.3\times$ (Opt1+2+3).

## VI. Related work

### A. Quantum Algorithm for Constrained Binary Optimization

The first quantum approach to this problem is quantum annealing [40], which uses the quantum adiabatic theorem to solve unconstrained binary optimization. This approach suffers from long embedding time and long evolution time [31]. To overcome this limit, QAOA provides a gate-model version of quantum annealing and uses a variation quantum algorithm to shorten evolution time. Since QAOA can be deployed on current noisy quantum devices, it has been studied and optimized from different aspects, such as compilation [3], parameter initialization [45], parameter updating [42], and distributed version [4]. However, quantum annealing and QAOA cannot deal with constraints. Although it can be extended by penalty term [44], cyclic Hamiltonian [47], they suffer from low success rates and long latency. This paper gives a confident and fast solution by integrating the commute Hamiltonian into QAOA and developing an efficient compilation flow.

Some universal quantum algorithms, alternating from quantum annealing and QAOA, are used to solve this problem with specialized modification. One is the hardware efficient ansatz (HEA) [13], [28]. Each variable is encoded by one qubit, and the objective is modified like penalty-QAOA. Although hardware-efficient, this method cannot always converge into an optimal solution since the circuit structure is not specialized. Another approach named Grover adaptive search uses the Grover algorithm [34], tailored with a selection circuit to exclude the solution outside the constraints [20]. However, the selection circuit is too complex to deploy on hardware. The search process generates too many solutions outside the constraints, requiring enormous iteration to find the target solution. Compared with these works, we use commute Hamiltonian to restrict the search space and multiple optimization strategies to reduce entanglement complexity and circuit depth.

### B. Unitary Decomposition for Hamiltonian Simulation

Hamiltonian simulation simulates the dynamics of a quantum system by decomposing the unitary into a quantum circuit composed of single and two-qubit gates. The most established method is Trotter-Suzuki decomposition [36], which uses numerous layers to decompose the total Hamiltonian unitary by repeating multiple local unitary on smaller subsystems. This method can obtain high-precision approximation but with a deep circuit. Other methods, like a linear combination [7] and quantum signal processing [35], use a quantum circuit to encode the sparse unitary for simulation. They can approximate Hamiltonian simulation with a shallow circuit but consume extra large ancillary qubit resources. Compared with these works, we provide a precise and efficient method for simulating commute Hamiltonian. We leverage the eigenspace feature of commute Hamiltonian and develop a precise decomposition with linear complexity and linear circuit depth.

## VII. Conclusion

This paper proposes Choco-Q, a fast and hardware-efficient approach for constrained binary optimization problems, which applies commute Hamiltonian that supports arbitrary linear constraints. Then, we develop three optimization techniques to reduce the circuit depth, including Hamiltonian serialization, equivalent decomposition, and variable elimination. These three techniques work together, shrinking the circuit depth from tens of thousands to hundreds. In conclusion, our design greatly improves the success rate and reduces the overall latency of QAOA.

## VIII. Acknowledgements

This work was supported by National Natural Science Foundation of China (No.62472374) and the National Key Research and Development Program of China (No. 2023YFF0905200). This work was also funded by Zhejiang Pioneer (Jianbing) Project (No. 2023C01036).

## IX. Artifact Appendix

### A. Abstract

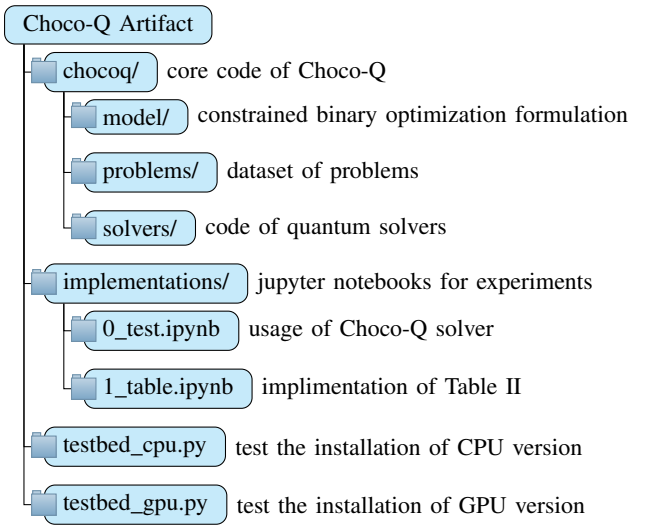
In this section, we provide detailed information that will facilitate the artifact evaluation process. The artifact check-list section presents brief information about this artifact and outlines the basic requirements to reproduce the experiment results. Then, we describe the directory tree of our source code and go into more detail about the requirements. Finally, in the experiment workflow section, we provide the usage example and how to reproduce the experiments.

### B. Artifact check-list (meta-information)

- **Algorithm:** Choco-Q, a formal and universal framework for constrained binary optimization problems. Choco-Q uses the simulation of commute Hamiltonian to embed constraints and achieve a 100% in-constraints rate to ensure high accuracy.
- **Program:** Choco-Q is implemented in Python.
- **Data set:** We use problems from 3 domains to evaluate Choco-Q, including facility location, graph coloring, and k-partition. We obtained specific forms of the objective function and constraints from the relevant literature [11], [26], [37], and faithfully constructed the relevant dataset in the framework of Choco-Q.
- **Run-time environment:** Ubuntu 20.04 LTS.
- **Hardware:** AMD EPYC 9554 64-core CPU, NVIDIA A100 Tensor Core GPU, 64GB Memory, 1TB of storage space.
- **Metrics:** In-constraints rate and success rate.
- **Results:** The results of experiments are shown in the output of the cells in the .ipynb file and are also written into the .csv file.
- **Experiments:** The jupyter notebooks are provided in `implimentations/` to reproduce the results of the experiments.
- **How much disk space is required (approximately)?:** 1TB.
- **How much time is needed to prepare workflow (approximately)?:** Less than ten minutes.
- **How much time is needed to complete experiments (approximately)?:** Less than two hours. For solving problems with higher problem scales, a GPU simulator will speed up the quantum circuit simulation, making the run time less than one hour. When using the CPU simulator, the quantum circuit simulation needs two hours to be completed.
- **Publicly available?:** Yes.
- **Code licenses (if publicly available)?:** GNU GPLv3.
- **Archived (provide DOI)?:** https://doi.org/10.5281/zenodo.14250941

### C. Description

Choco-Q is built on Python scripts, which can be executed on Linux systems. Below, we introduce the important files and directories in the artifact.




- **chocoq/.** This sub-directory contains the core code of Choco-Q, which includes the constrained binary optimization model (`model/`), the problem dataset (`problems/`), the implementation of Choco-Q solver and other baseline solvers (`solvers/`).
- **implimentations/.** This sub-directory includes the Jupyter Notebooks to reproduce the experiments in the paper.
- **testbed_cpu.py.** This Python script gives an example using Choco-Q to define the optimization problem and solve it with Choco-Q with CPU simulator.
- **testbed_gpu.py.** This Python script gives an example using Choco-Q to define the optimization problem and solve it with Choco-Q with a GPU simulator.

**How to access:** DOI: 10.5281/zenodo.14250941
GitHub: https://github.com/JanusQ/Choco-Q

**Hardware dependencies:** The evaluation in the paper is performed on a server with AMD EPYC 9554 64-core Processor, 1511GB memory, and 32TB storage space. Running the code requires an x86-64 machine with at least 64GB memory and 1TB storage space.

**Software dependencies:** The code relies on Python 3.10 or higher version. We list all required packages in `environment_cpu.yml` of the artifact.

### D. Dataset

The data used has been provided in `chocoq/problems`, which includes the problems in experiments.

### E. Installation

1) Download the source code from the GitHub repository (https://github.com/JanusQ/Choco-Q.git) and enter into the `chocoq/` directory.
2) Install Anaconda or Miniconda and create a virtual environment from the configuration file.

```
conda env create -f environment_cpu.yml
```

3) Activate the virtual environment and install chocoq.

```
conda activate choco_cpu
pip install .
```

If you want to install the GPU version, please replace the `cpu` in the previous commands with `gpu`.

After finishing the above installation, you can run the `testbed_cpu.py` to check the installation of the CPU version.

```
python testbed_cpu.py
```

To test the installation of the GPU version, run

```
python testbed_gpu.py
```

Please refer to the `README.md` in the artifact for a more detailed description.

### F. Using Choco-Q

Users can 1) *define a constrained binary optimization problem* and 2) *solve the problem with Choco-Q solver or other quantum solvers*. We also provide an example code in `testbed_cpu.py` to illustrate the usage.

### G. Reproducing Experimental Results

- **Jupyter Notebooks.** The `1_table.ipynb` in directory `implimentations/` provides the Jupyter Notebook to reproduce the experimental results in Table II. You can run it to reproduce the experimental results.

### H. Methodology

Submission, reviewing, and badging methodology:

- https://www.acm.org/publications/policies/artifact-review-badging
- http://cTuning.org/ae/submission-20201122.html
- http://cTuning.org/ae/reviewing-20201122.html

## REFERENCES


[1] S. Akibue, G. Kato, and S. Tani, "Probabilistic unitary synthesis with optimal accuracy," *ACM Transactions on Quantum Computing (TQC)*, 2023.

[2] M. Alam, A. Ash-Saki, and S. Ghosh, "Circuit compilation methodologies for quantum approximate optimization algorithm," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2020, pp. 215–228.

[3] M. Alam, A. Ash-Saki, and S. Ghosh, "An efficient circuit compilation flow for quantum approximate optimization algorithm," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*, 2020, pp. 1–6.

[4] R. Ayanzadeh, N. Alavisamani, P. Das, and M. Qureshi, "Frozenqubits: Boosting fidelity of qaoa by skipping hotspot nodes," in *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2023, pp. 311–324.

[5] M. Ayodele, "Penalty weights in qubo formulations: Permutation problems," in *European Conference on Evolutionary Computation in Combinatorial Optimization (EvoStar)*, 2022, pp. 159–174.

[6] S. Benati and R. Rizzi, "A mixed integer linear programming formulation of the optimal mean/value-at-risk portfolio problem," *European Journal of Operational Research*, vol. 176, no. 1, pp. 423–434, 2007.

[7] D. W. Berry, A. M. Childs, and R. Kothari, "Hamiltonian simulation with nearly optimal dependence on all parameters," in *2015 IEEE 56th Annual Symposium on Foundations of Computer Science (FOCS)*, 2015, pp. 792–809.

[8] L. Bittel and M. Kliesch, "Training variational quantum algorithms is np-hard," *Physical Review Letters*, vol. 127, no. 12, p. 120502, 2021.

[9] A. Bocharov, M. Roetteler, and K. M. Svore, "Efficient synthesis of universal repeat-until-success quantum circuits," *Physical Review Letters*, vol. 114, no. 8, p. 080502, 2015.

[10] 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 Information Processing*, vol. 22, no. 1, p. 25, 2022.

[11] T. N. Bui and B. R. Moon, "Genetic algorithm and graph partitioning," *IEEE Transactions on Computers (TC)*, vol. 45, no. 7, pp. 841–855, 1996.

[12] C. Chen, B. Schmitt, H. Zhang, L. S. Bishop, and A. Javadi-Abhar, "Optimizing quantum circuit synthesis for permutations using recursion," in *Proceedings of the 59th ACM/IEEE Design Automation Conference (DAC)*, 2022, pp. 7–12.

[13] S. Dangwal, G. S. Ravi, P. Das, K. N. Smith, J. M. Baker, and F. T. Chong, "Varsaw: Application-tailored measurement error mitigation for variational quantum algorithms," in *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2023, pp. 362–377.

[14] M. A. de Gosson, "Born–jordan quantization and the equivalence of the schrödinger and heisenberg pictures," *Foundations of Physics*, vol. 44, no. 10, pp. 1096–1106, 2014.

[15] I. Q. Devices, "ibmq quito," 2024. [Online]. Available: https://quantum-computing.ibm.com/services/resources

[16] D. Dugošija, A. Savić, and Z. Maksimović, "A new integer linear programming formulation for the problem of political districting," *Annals of Operations Research*, vol. 288, pp. 247–263, 2020.

[17] R. Z. Farahani and M. Hekmatfar, *Facility location: concepts, models, algorithms and case studies*. Springer Science & Business Media, 2009.

[18] E. Farhi, J. Goldstone, and S. Gutmann, "A quantum approximate optimization algorithm," *arXiv preprint arXiv:1411.4028*, 2014.

[19] C. Gidney, "Constructing large controlled nots," 2015. [Online]. Available: https://algassert.com/circuits/2015/06/05/Constructing-Large-Controlled-Nots.html

[20] A. Gilliam, S. Woerner, and C. Gonciulea, "Grover adaptive search for constrained polynomial binary optimization," *Quantum*, vol. 5, p. 428, 2021.

[21] Gurobi Optimization, LLC, "Gurobi Optimizer Reference Manual," 2024. [Online]. Available: https://www.gurobi.com

[22] S. Hadfield, Z. Wang, B. O'gorman, E. G. Rieffel, D. Venturelli, and R. Biswas, "From the quantum approximate optimization algorithm to a quantum alternating operator ansatz," *Algorithms*, vol. 12, no. 2, p. 34, 2019.

[23] S. Hadfield, Z. Wang, E. G. Rieffel, B. O'Gorman, D. Venturelli, and R. Biswas, "Quantum approximate optimization with hard and soft constraints," in *Proceedings of the Second International Workshop on Post Moores Era Supercomputing (PMES)*, 2017, pp. 15–21.

[24] W. Heisenberg, *The physical principles of the quantum theory*. Courier Corporation, 1949.

[25] W. Herroelen, "Project scheduling—theory and practice," *Production and Operations Management*, vol. 14, no. 4, pp. 413–432, 2005.

[26] T. R. Jensen and B. Toft, *Graph coloring problems*. John Wiley & Sons, 2011.

[27] T. Kadowaki and H. Nishimori, "Quantum annealing in the transverse ising model," *Physical Review E*, vol. 58, no. 5, p. 5355, 1998.

[28] A. Kandala, A. Mezzacapo, K. Temme, M. Takita, M. Brink, J. M. Chow, and J. M. Gambetta, "Hardware-efficient variational quantum eigensolver for small molecules and quantum magnets," *Nature*, vol. 549, no. 7671, pp. 242–246, 2017.

[29] R. M. Karp, *Reducibility among combinatorial problems*. Springer, 2010.

[30] D. Knop, M. Pilipczuk, and M. Wrochna, "Tight complexity lower bounds for integer linear programming with few constraints," *ACM Transactions on Computation Theory (TCT)*, vol. 12, no. 3, pp. 1–19, 2020.

[31] C. R. Laumann, R. Moessner, A. Scardicchio, and S. L. Sondhi, "Quantum annealing: The fastest route to quantum computation?" *The European Physical Journal Special Topics*, vol. 224, no. 1, pp. 75–88, 2015.

[32] H. Leipold and F. M. Spedalieri, "Constructing driver hamiltonians for optimization problems with linear constraints," *Quantum Science and Technology*, vol. 7, no. 1, p. 015013, 2021.

[33] G. Li, Y. Ding, and Y. Xie, "Tackling the qubit mapping problem for nisq-era quantum devices," in *Proceedings of the 24th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2019, pp. 1001–1014.

[34] G.-L. Long, "Grover algorithm with zero theoretical failure rate," *Physical Review A*, vol. 64, no. 2, p. 022307, 2001.
[35] G. H. Low and I. L. Chuang, "Optimal hamiltonian simulation by quantum signal processing," *Physical Review Letters*, vol. 118, no. 1, p. 010501, 2017.
[36] D. Mac Kernan, G. Ciccotti, and R. Kapral, "Trotter-based simulation of quantum-classical dynamics," *The Journal of Physical Chemistry B*, vol. 112, no. 2, pp. 424–432, 2008.
[37] M. T. Melo, S. Nickel, and F. Saldanha-Da-Gama, "Facility location and supply chain management–a review," *European Journal of Operational Research*, vol. 196, no. 2, pp. 401–412, 2009.
[38] A. Omu, R. Choudhary, and A. Boies, "Distributed energy resource system optimisation using mixed integer linear programming," *Energy Policy*, vol. 61, pp. 249–266, 2013.
[39] M. J. Powell, *A direct search optimization method that models the objective and constraint functions by linear interpolation*. Springer, 1994.
[40] G. E. Santoro and E. Tosatti, "Optimization using quantum mechanics: quantum annealing through adiabatic evolution," *Journal of Physics A: Mathematical and General*, vol. 39, no. 36, p. R393, 2006.
[41] S. Stein, N. Wiebe, Y. Ding, P. Bo, K. Kowalski, N. Baker, J. Ang, and A. Li, "Eqc: ensembled quantum computing for variational quantum algorithms," in *Proceedings of the 49th Annual International Symposium on Computer Architecture (ISCA)*, 2022, pp. 59–71.
[42] M. Streif and M. Leib, "Training the quantum approximate optimization algorithm without access to a quantum processing unit," *Quantum Science and Technology*, vol. 5, no. 3, p. 034008, 2020.
[43] S. Tan, C. Lang, L. Xiang, S. Wang, X. Jia, Z. Tan, T. Li, J. Yin, Y. Shang, A. Python, L. Lu, and J. Yin, "Quct: A framework for analyzing quantum circuit by extracting contextual and topological features," in *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2023, pp. 494–508.
[44] A. Verma and M. Lewis, "Penalty and partitioning techniques to improve performance of qubo solvers," *Discrete Optimization*, vol. 44, p. 100594, 2022.
[45] M. Wang, B. Fang, A. Li, and P. J. Nair, "Red-qaoa: Efficient variational optimization through circuit reduction," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2024, pp. 980–998.
[46] A. Xu, A. Molavi, L. Pick, S. Tannu, and A. Albarghouthi, "Synthesizing quantum-circuit optimizers," *Proceedings of the ACM on Programming Languages (PLDI)*, pp. 835–859, 2023.
[47] T. Yoshioka, K. Sasada, Y. Nakano, and K. Fujii, "Experimental demonstration of fermionic qaoa with one-dimensional cyclic driver hamiltonian," in *2023 IEEE International Conference on Quantum Computing and Engineering (QCE)*, vol. 1, 2023, pp. 300–306.