Received 11 July 2023, accepted 17 September 2023, date of publication 22 September 2023, date of current version 27 September 2023.

*Digital Object Identifier 10.1109/ACCESS.2023.3318206*

RESEARCH ARTICLE

# Classifying and Benchmarking Quantum Annealing Algorithms Based on Quadratic Unconstrained Binary Optimization for Solving NP-Hard Problems

**JEHN-RUEY JIANG, (Member, IEEE), AND CHUN-WEI CHU**
Department of Computer Science and Information Engineering, National Central University, Taoyuan 320317, Taiwan

Corresponding author: Jehn-Ruey Jiang (jrjiang@csie.ncu.edu.tw)

This work was supported by the National Central University, Taiwan.

**ABSTRACT** Quantum annealing has the potential to outperform classical transistor-based computer technologies in tackling intricate combinatorial optimization problems. However, ongoing scientific debates cast doubts on whether quantum annealing devices (or quantum annealers) can genuinely provide better problem-solving capabilities than classical computers. The question of whether quantum annealing algorithms (QAAs) running on quantum annealers have computational advantages over classical algorithms (CAs) running on classical computers still remains unclear. This paper aims to clarify the question by classifying and benchmarking QAAs that utilize quadratic unconstrained binary optimization (QUBO) formulas to solve NP-hard problems. It proposes a four-class classification of QUBO formulas and exemplifies each class by QUBO formulas used by QAAs for solving specific NP-hard problems, such as the subset sum, maximum cut, vertex cover, 0/1 knapsack, graph coloring, Hamiltonian cycle, traveling salesperson, and job shop scheduling problems. The classification is based on the following two criteria: (i) Does the number of QUBO variables scale linearly with the problem input size? (ii) Does the QUBO formula have both the constraint term and the optimization term? QAAs are implemented and run on a D-Wave quantum annealer for benchmarking. They are benchmarked against related CAs in terms of the quality of the solution and the time to the solution. The benchmarking results reveal which classes of QUBO formulas are likely to provide advantages to QAAs over CAs. Furthermore, based on the benchmarking results, observations and suggestions are given for each class of QUBO formulas, facilitating the adoption of appropriate actions to improve the performance of QAAs.

**INDEX TERMS** Noisy intermediate-scale quantum, NP-hard problem, quantum annealing, quantum computer, quadratic unconstrained binary optimization.

## I. INTRODUCTION

Quantum computers perform computation based on quantum bits or qubits with the phenomena of quantum superposition, quantum entanglement, quantum tunneling, and so on. A qubit exists in a superposition of both 0 and 1, and definitely reveals 0 or 1 when measured. In contrast, classical computers perform computation based on bits, each of which is either 0 or 1. Quantum computers have been attracting much research attention because they can offer computing power that classical computers can never offer, which is called quantum supremacy [1].

Some quantum computers, such as IBM Q [2] Google Sycamore [3], are universal, whereas some others, such as D-Wave Advantage [4], are non-universal. On the one hand, universal quantum computers rely on quantum gates to form quantum circuits to perform computation to solve general problems. On the other hand, other mechanisms rather than quantum gates are employed to perform computation for solving special problems on non-universal quantum computers.

The associate editor coordinating the review of this manuscript and approving it for publication was Wei Huang.

For example, the quantum annealing mechanism is used by D-Wave Advantage, a quantum annealer, to leverage quantum tunneling to traverse the energy landscape to find the globally lowest energy [5] for solving optimization problems.

Quantum annealers hold the promise of outperforming classical computers in solving intricate combinatorial optimization problems arising in different application areas, such as quantum chemistry [6], quantum machine learning [7], [8], [9], quantum deep learning [10], [11], quantum variational autoencoders [12], [13], fault detection and diagnosis [14], [15], online fraud detection [16], financial portfolio optimization [17], [18], operational planning [19], [20], data processing in high energy physics [21], [22], material microstructure equilibration [23], [24] and Monte Carlo sampling [25]. However, as mentioned in [26], there are ongoing scientific debates [27], [28], [29], [30] arguing whether quantum annealers can provide better problem solving capabilities than classical computers. The question still remains unclear regarding whether quantum annealing algorithms (QAAs) running on quantum annealers have computational advantages over classical algorithms (CAs) running on classical computers. We are thus motivated to write this paper aiming to clarify the question by classifying and benchmarking QAAs that use quadratic unconstrained binary optimization (QUBO) formulas to solve NP-hard problems.

This paper classifies QUBO formulas into four classes, as previously demonstrated in [31]. It also exemplifies each class with two QUBO formulas used by two QAAs for solving specific NP-hard problems. The solved problems are the subset sum problem (SSP), maximum cut problem (MCP), vertex cover problem (VCP), 0/1 knapsack problem (0/1 KP), graph coloring problem (GCP), Hamiltonian cycle problem (HCP), traveling salesperson problem (TSP), and job shop scheduling problem (JSP). It is believed that no CA can solve an NP-hard problem efficiently with a polynomial time complexity in the worst case. That is to say, for an NP-hard problem, the best CA to solve it needs a super-polynomial (e.g., exponential) time complexity in the worst case. The QAAs based on QUBO to solve the above-mentioned NP-hard problems are implemented and run on a D-Wave quantum annealer. They are compared with related CAs for benchmarking in terms of the quality of the solution and the time to the solution. The benchmarking results reveal which classes of QUBOs are likely to provide advantages to QAAs over CAs. Based on the benchmarking results, observations and suggestions are given for each QUBO formula class, so that proper actions can be adopted to improve the performance of QAAs. Compared with CAs, QAAs using QUBO formulas are competitive in the current NISQ era [32], in which quantum computers have only a moderate number of error-prone qubits with low-fidelity. It is believed many QAAs will have superior performance to CAs in the future when we go beyond the NISQ era to have quantum computers owning thousands or more qubits with high fidelity.

The contribution of this paper is fourfold. First, it proposes a four-class classification of QUBO formulas used by QAAs for solving NP-hard problems. It also exemplifies each class by QUBO formulas solving specific NP-hard problems. Second, QAAs are implemented and run on a D-Wave quantum annealer to solve the specific NP-hard problems. They are compared with related CAs for benchmarking in terms of the quality of the solution and the time to the solution. Third, this paper identifies which classes of QUBO formulas are likely to provide advantages to QAAs over CAs. Fourth, observations and suggestions are given for each QUBO formula class, so that appropriate actions can be taken to improve the performance of QAAs. Generally, the significance of this paper is (i) for people to determine if a QAA using a QUBO formula to solve a certain NP-hard problem is likely to outperform related CAs by checking the QUBO formula class, and (ii) for people to take proper actions to further improve the QAA performance.

The remainder of this paper is organized as follows. Section II introduces some preliminaries. The QUBO formula classification is described in Section III. QUBO formulas used by QAAs solving specific NP-hard problems are shown as examples of classification classes in Section IV. For the purpose of benchmarking, the QAAs are also compared with related CAs in terms of different performance metrics in Section V. Observations and suggestions based on QAA benchmarks are given in Section VI. Finally, Section VII concludes this paper.

## II. PRELIMINARIES

### A. NP-HARD PROBLEMS

In this subsection, we introduce the concept of NP-hard problems. As mentioned earlier, it is well believed that no CAs can solve NP-hard problems efficiently with a polynomial time complexity for any problem instances. Below, we first introduce the concepts of deterministic algorithms and nondeterministic algorithms [33] for realizing NP-hard problems.

The deterministic algorithm is the normal CA that can run on current classical general-purpose computers to find solutions to problems. On the contrary, according to [34], the nondeterministic algorithm cannot run on any current computers; it is conceptual and intended only for theoretical discussions. A nondeterministic algorithm to solve a given problem has two phases: choosing and checking. The choosing phase is to select one out of given options. The checking phase is to check whether the selected option leads to a correct solution to the problem or not. If so, it returns "success", which means that a correct solution can be found; otherwise, it returns "failure", which means that no correct solution can be found. The nondeterministic algorithm has a strong assumption that if there exist proper options leading to correct solutions, then the choosing phase is assumed to always select one of the proper options for the algorithm to return "success".

On the one hand, problems that can be solved by the deterministic algorithm with a polynomial time complexity constitute a problem set called P; they are called P problems. On the other hand, problems that can be solved by the nondeterministic algorithm with a polynomial time complexity constitute a problem set called NP; they are called NP problems. It is believed, but has not yet been proven, that P is a proper subset of NP.

Cook proved that every NP problem can be "polynomially reducible to" the satisfiability (SAT) problem [34]. A problem X is said to be "polynomially reducible to" a problem Y if and only if X can be solved by (i) converting X's input instance into Y's input instance with a polynomial time complexity, (ii) getting an algorithm to solve the problem Y with the converted input instance to output Y's solution, and (iii) converting Y's solution into X's solution with a polynomial time complexity. According to Cook's proof, if the SAT problem belongs to P, then all NP problems also belong to P. A problem is called an NP-hard problem if every NP problem is polynomially reducible to it. Therefore, the SAT problem is an NP-hard problem. Many problems have been shown to be NP-hard problems. For example, Karp introduced 21 NP-hard problems, such as the SSP, MCP, VCP, 0/1 KP, HCP, and TSP [35]. Up to now, no NP-hard problem has been solved by any deterministic algorithm with a polynomial time complexity in the worst case. And it is believed that there will be no such algorithm. Thus, we may well say that NP-hard problems are very hard problems to solve.

## *B. QUANTUM ANNEALING AND QUBO FORMULAS*

Quantum annealing (QA) is a mechanism [36] leveraging quantum tunneling to solve optimization problems that optimize objective functions of very large solution spaces. Quantum tunneling [37] is a quantum mechanics phenomenon about energy levels. When a quantum particle encounters an energy barrier, it may pass through the barrier even if its momentum is less than the barrier potential.

For an objective function, QA first prepares states associated with the function. It then initiates an adiabatic process starting from a quantum superposition of all possible candidate states of the solution space with equal probability amplitudes. Afterwards, the amplitudes of all states keep changing at the same time, which is like traversing all states in parallel. If the changing rate is slow enough, then the adiabatic process stops with a state close to the ground state of the lowest Hamiltonian (or total system energy) associated with the objective function, which corresponds to the optimal solution. In summary, with the quantum tunneling phenomenon, the QA mechanism behaves as traversing the whole solution space in parallel to find the globally optimal solution to the objective function. The solution found does not get stuck in the local optimization (or minimum), but reaches the global optimization (or minimum), as shown in Figure 1.

Based on the QA mechanism, a QAA formulates an optimization problem as an objective function of a quadratic unconstrained binary optimization (QUBO) formula $f$ of binary variables, as described in the following Equation (1).

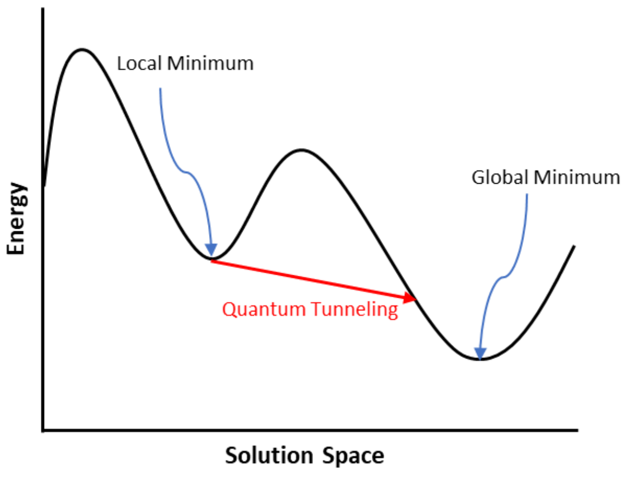


**FIGURE 1. Illustration of quantum annealing that leverages quantum tunneling to find the global minimum in the solution space.**

$$f(x) = x^T Qx = \sum_i Q_{i,i}x_i^2 + \sum_{i<j} Q_{i,j}x_ix_j, \tag{1}$$

where $Q$ is an $n \times n$ upper triangular matrix with real-number coefficients, and $x = (x_1\, x_2\, \ldots\, x_n)^T$ is a column vector of $n$ binary variables of either value 0 or 1. The minimum value of the objective function corresponds to the optimal solution to the problem. Note that $x_i$ is either 0 or 1, so Equation (1) can be rewritten as Equation (2) shown below. Also note that Equation (1) and Equation (2) are true for the Ising formula of variables of either value -1 or 1, which is a well-known model equivalent to the QUBO formula.

$$f(x) = \sum_i Q_{i,i}x_i + \sum_{i<j} Q_{i,j}x_ix_j, \tag{2}$$

A QUBO formula should have no constraint term, as suggested by its name. However, some optimization problems have constraints of feasible solutions. Any equality constraint $Rx = t$ can be transformed into a constraint term or penalty term of the form $\alpha(Rx - t)^2$, where $R$ is a $1 \times n$ matrix (or row vector) with real-number coefficients, $x$ is a column vector of variables with binary values, $t$ is a constant, and $\alpha$ is called a constraint weight or penalty weight of a real-number value. By integrating the penalty term $\alpha(Rx - t)^2$ and the optimization term $x^T Qx$, the objective function $f(x)$ is extended to be the formula shown in Equation (3) below.

$$f(x) = \alpha(Rx - t)^2 + x^T Qx \tag{3}$$

Equation (3) sometimes is extended to be a more general formula, as shown in Equation (4) below.

$$f(x) = \alpha(Rx - t)^2 + \beta x^T Qx, \tag{4}$$

where $\beta$ is called the optimization weight of a real-number value associated with the optimization term $x^T Qx$.

The value of weights $\alpha$ and $\beta$ should be set properly so that the whole QUBO objective function can be minimized to return an optimal and feasible solution to the problem. Some weight setting methods are proposed to set the weights properly. They are elaborated in the next subsection.

### C. SETTING QUBO PENALTY WEIGHTS

In this subsection, we elaborate weight setting methods (WSMs) [38], [39] to properly set the penalty weight $\alpha$ of the QUBO formula shown in the following Equation (5).

$$f(x) = \alpha(Rx - t)^2 + x^T Qx = \alpha g(x) + c(x), \qquad (5)$$

where $\alpha$ is the penalty weight with a real-number value, $R$ is a row vector with real-number coefficients, $x$ is a column vector of binary variables with either value 0 or 1, $t$ is a constant, $Q$ is an upper triangular matrix with real-number coefficients, $g(x)$ is the constraint function, and $c(x)$ is the cost function or the optimization function.

In Equation (5), $g(x) > 0$ if $x$ represents an infeasible solution, whereas $g(x) = 0$ if $x$ represents a feasible solution. Let $y$ be the optimal solution that makes $f(y)$ have the minimal value, and $S$ be the space of all infeasible solutions. We have the following inequality:

$$c(y) < \alpha g(x) + c(x), \text{ for every } x \in S. \qquad (6)$$

According to Equation (6), a valid penalty weight $\alpha$ must satisfy the following inequality:

$$\alpha > \max_{x \in S} \left( \frac{c(y) - c(x)}{g(x)} \right). \qquad (7)$$

Based on Equation (7), different WSMs [38], [39] are proposed to set $\alpha$ as various values. The methods are Verma and Lewis Method (VLM), Upper Bound (UB), Maximum QUBO Coefficient (MQC), Maximum change in Objective function divided by Minimum Constraint function of infeasible solutions (MOMC), and Maximum value derived from dividing each change in Objective function with the corresponding change in Constraint function (MOC).

Table 1 shows the penalty weights associated with the above-mentioned WSMs. In the table, $G$ and $C$ are $n \times n$ matrices representing $g(x)$ and $c(x)$, respectively. Moreover, $z$ is a column vector with all 1's, so $\alpha_{UB}$ is an upper bound of the objective function with all positive QUBO formula coefficients. $\alpha_{MQC}$ is the maximum QUBO formula coefficient. $\alpha_{VLM}$ is a good estimation of the numerator (i.e., $c(y) - c(x)$) of Equation (7) without considering the denominator (i.e., $g(x)$). $\alpha_{MOMC}$ considers $g(x)$ to improve $\alpha_{VLM}$ by estimating $g(x)$ as $\gamma$, the minimum change in the constraint function that is larger than 0. $\alpha_{MOC}$ tries to further improve $\alpha_{VLM}$ by considering a possible increase in the constraint function as a result of a change in the objective function which can be achieved by flipping any bit from 0 to 1 or vice versa. The readers are referred to [38] and [39] for details of setting the penalty weight $\alpha$.

### D. RUNNING A QAA ON A QUANTUM ANNEALER

Figure 2 shows the five major steps to design and run a QAA for solving an optimization problem on a quantum annealer, such as the D-Wave Advantage quantum computer. The five steps are elaborated below.

Step 1. Problem formulation (or definition): The optimization problem is formulated as the QUBO formula of a QAA. Note that some studies formulate the problem as the Ising model [40]. For example, the paper [41] formulates 21 NP-hard problems as Ising models. It has been shown in [42] that the QUBO formula and the Ising model are equivalent, and can be transformed to each other easily. Since this paper focuses on the QUBO formula, an optimization problem is formulated as a QUBO formula in the following context. The formula is in turn transformed into a graph in which a node (or vertex) stands for a binary variable, and the weight of an edge between two nodes represents the coupling strength between the two variables associated with the two nodes.

Step 2. Minor embedding: The graph corresponding to the QUBO formula is embedded in the quantum processor or quantum processing unit (QPU) of the quantum annealer, with qubits and couplers representing graph nodes and edges, respectively, as shown in Figure 2. Ideally, a qubit should be directly connected to every qubit that has a coupling relationship with it. However, due to the hardware limitation of qubit connectivities, a qubit can only be directly connected to a certain number of qubits. For example, a qubit can be directly connected to 6 and 15 other qubits in the D-wave quantum annealer Chimera and Pegasus architectures, respectively.

In order to maintain the coupling relationships of nodes in the graph, multiple qubits are used to represent a node, and couplers (or chains) of strong strength are set between each other of these qubits to make them maintain the same value. Note that a doubled line between two nodes of some graphs in Figure 2 represents a chain.

When the required number of qubits exceeds the upper limit of the device, the graph corresponding to the original problem should be decomposed into subgraphs to be properly embedded into the QPU. Currently known graph decomposition (or problem decomposition) methods include the iterative centrality halo method [43], which prioritizes nodes that have significant impacts on the global solution, and the DBK (Decomposition, Bounds, K-core) method, which recursively decomposes a graph into subgraphs of specific sizes [44]. Certainly, the performance of quantum annealing is greatly affected by graph decomposition methods [43], [44].

Step 3. Initialization: This step sets the initial Hamiltonian $H_i$ and the final Hamiltonian $H_f$ of the entire system. For a particular state of the system, the Hamiltonian represents the total energy of the system in that state. The initial Hamiltonian $H_i$ is set to make every qubit stay in a superposition state. The final Hamiltonian $H_f$ is set according to the QUBO formula, so that the minimum final Hamiltonian corresponds to the optimal solution to the problem.

**TABLE 1. Different weight setting methods (WSMs) and their associated penalty weights.**

| WSMs | Weights |
|---|---|
| UB | $\alpha_{UB} = z^T C z, z_i = 1, i = 1, \ldots, n$ |
| MQC | $\alpha_{MQC} = \max_{i=1}^{n} \max_{j=1}^{n} C_{i,j}$ |
| VLM | $\alpha_{VLM} = \max_{i=1}^{n} W_i^c$, where $W_i^c = \max\left(-C_{i,i} - \sum_{j=1, j\neq i}^{n} \min(C_{i,j}, 0),\ \ C_{i,i} + \sum_{j=1, j\neq i}^{n} \max(C_{i,j}, 0)\right)$ |
| MOMC | $\alpha_{MOMC} = \max\left(1, \frac{\alpha_{VLM}}{\gamma}\right)$, where $\gamma = \min_{i=1, W_i^g>0}^{n} W_i^g$, and $W_i^g = \min\left(-G_{i,i} - \sum_{j=1, j\neq i}^{n} \min(G_{i,j}, 0),\ \ G_{i,i} + \sum_{j=1, j\neq i}^{n} \max(G_{i,j}, 0)\right)$ |
| MOC | $\alpha_{MOC} = \max\left(1, \max_{i=1, W_i^g>0}^{n} \left\lvert \frac{W_i^c}{W_i^g} \right\rvert\right)$ |

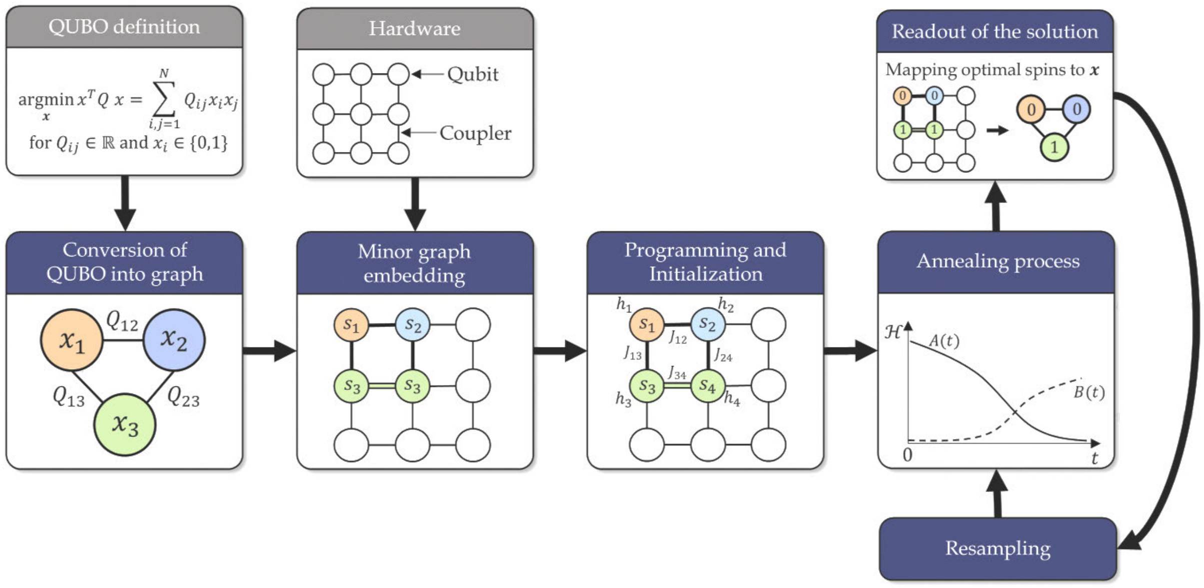

QUBO definition
$\arg\min_x x^T Q x = \sum_{i,j=1}^{N} Q_{ij} x_i x_j$
for $Q_{ij} \in \mathbb{R}$ and $x_i \in \{0,1\}$
Hardware
Qubit
Coupler
Readout of the solution
Mapping optimal spins to $x$
Conversion of QUBO into graph
$x_1$ $x_2$ $x_3$ $Q_{12}$ $Q_{13}$ $Q_{23}$
Minor graph embedding
$s_1$ $s_2$ $s_3$
Programming and Initialization
$h_1$ $h_2$ $h_3$ $h_4$ $J_{12}$ $J_{13}$ $J_{24}$ $J_{34}$ $s_4$
Annealing process
$\mathcal{H}$ $A(t)$ $B(t)$ 0 $t$
Resampling

**FIGURE 2. The workflow of running a QAA on a D-Wave quantum annealer (adapted from [36]).**

The system Hamiltonian $H(t)$ at time $t$ of a quantum annealer can be expressed as the following Equation (8).

$$H(t) = A(t)H_i + B(t)H_f, \tag{8}$$

where $A(t)$ and $B(t)$ are Hamiltonian scaling functions that evolve with the annealing time $t$. On the one hand, $A(t)$ gradually goes from 1 to close to 0; on the other hand, $B(t)$ gradually goes from 0 to close to 1.

Step 4. Annealing: In this step, the annealing process is performed to obtain the optimal (or minimum) value of the objective function. The system starts from the lowest initial Hamiltonian, where every qubit is in a superposition state. Then during annealing, the initial Hamiltonian decreases gradually, whereas the final Hamiltonian increases gradually. Finally, at the end of the annealing, the effect of the initial Hamiltonian drops to zero, and the system is in the lowest energy state of the final Hamiltonian associated with the QUBO formula of the objective function. Each qubit collapses from the superposition state to the state of 0 or 1, which corresponds to the binary variable value achieving the final global optimal objective function value.

The lower right part of Figure 2 shows the change in the energy scaling functions $A(t)$ and $B(t)$ in Equation (8). During the annealing process, the energy scaling function $A(t)$ and $B(t)$ gradually grow smaller and larger with time $t$, respectively. Thus, the influence of the initial Hamiltonian $H_i$ becomes more significant, whereas the influence of the final Hamiltonian $H_f$ becomes less significant. And at the end of annealing, the system Hamiltonian is mostly reflected by the final Hamiltonian $H_f$.

Step 5. Reading: After the annealing process, the value (0 or 1) of each qubit is read out for post-processing. According to the corresponding relationship between the qubit and the graph node, a solution to the original problem can be derived. If the values of qubits representing the same node are inconsistent, post-processing mechanisms, such as the majority vote, are employed to determine what the value of the node is. Note that Steps 4 and 5 are repeated many times (shots), which is called the resampling process, to ensure that the optimal solution to the problem can be found with high probability.

## III. QAAS SOLVING NP-HARD PROBLEMS

This section presents QAAs using QUBO formulas to solve specific NP-hard problems, including the SSP, MCP, VCP, 0/1 KP, GCP, HCP, TSP, and JSP. The QAAs are elaborated one by one in the following subsections.

### A. THE QAA SOLVING THE SSP

The subset sum problem (SSP) is defined as follows:

Given a set $S = \{s_1, s_2, \ldots, s_n\}$ with $n$ integers, and a target integer $T$, the SSP is to find a subset $S'$ of $S$ such that the sum of the integers in the subset $S'$ is exactly $T$.

The QAA solving the SSP utilizes the following QUBO formula:

$$H(x) = \left( \sum_{i=1}^{n} s_i x_i - T \right)^2 \tag{9}$$

In Equation (9), $s_i$ is an integer in $S$, $x_i = 1$ represents that $s_i$ is in $S'$, and $x_i = 0$ represents that $s_i$ is not in $S'$, where $1 \leq i \leq n$. The number of QUBO variables is of O($n$), which scales linearly with the problem size $n$, and the QUBO formula has only the constraint term.

### B. THE QAA SOLVING THE MCP

The maximum cut problem (MCP) is defined as follows:

Given an undirected graph $G = (V, E)$ with the vertex set $V$ and the edge set $E$, a cut in $G$ is a subset $S \in V$. The MCP is to find a cut $S$ such that $EWS(S, S')$ is maximized, where $S' = V - S$, and $EWS(S, S')$ stands for the edge weight sum of edges between $S$ and $S'$.

The QAA solving the MCP utilizes the following QUBO formula:

$$H(x) = \sum_{(u,v)\in E} w_{uv}(-x_u - x_v + 2x_u x_v) \tag{10}$$

In Equation (10), $x_u = 1$ (resp., $x_u = 0$) indicates that $u$ is (resp., is not) in $S$, $x_v = 1$ (resp., $x_v = 0$) indicates that $v$ is (resp., is not) in $S$, and $w_{uv}$ is the weight associated with the edge $(u, v)$. The number of QUBO variables is of O($n$), which scales linearly with the problem size $n$, and the QUBO formula has only the optimization term.

### C. THE QAA SOLVING THE VCP

The vertex cover problem (VCP) is defined as follows:

Given an undirected graph $G = (V, E)$ with the vertex set $V$ and the edge set $E$, the VCP is to find a minimum-sized subset $V'$ of $V$, such that for every edge $(u, v)$ in $E$, either $u$ or $v$ is in $V'$.

The QAA solving the VCP utilizes the following QUBO formula:

$$H(x) = A \sum_{(u,v)\in E} (1 - x_u)(1 - x_v) + B \sum_{v\in V} x_v \tag{11}$$

In Equation (11), $x_u = 1$ (resp., $x_u = 0$) indicates that $u$ is (resp., is not) in $V'$, and $x_v = 1$ (resp., $x_v = 0$) indicates that $v$ is (resp., is not) in $V'$. The first term is the constraint term whose weight is $A$, whereas the second term is the optimization term whose weight is $B$. The number of QUBO variables is of O($n$), which scales linearly with the problem size $n$, and the QUBO formula has both the constraint term and the optimization term.

### D. THE QAA SOLVING THE 0/1 KP

The 0/1 knapsack problem (0/1 KP) is defined as follows:

Consider a knapsack with capacity $W$ and $n$ objects $o_1, \ldots, o_n$ whose weights are $w_1, \ldots, w_n$ and whose costs are $c_1, \ldots, c_n$. The 0/1 KP is to select objects to form a set $S$ such that $\sum_{o_i \in S} c_i$ is maximized under the constraint $\sum_{o_i \in S} w_i \leq W$ (i.e., selected objects can be accommodated by the knapsack and have the maximum total cost).

The QAA solving the 0/1 KP utilizes the following QUBO formula:

$$\begin{aligned} H(x) = A &\left(1 - \sum_{j}^{W} y_j\right)^2 \\ &+ A \left( \sum_{j=1}^{W} j y_j - \sum_{i=1}^{n} w_i x_i \right)^2 \\ &- B \sum_{i=1}^{n} c_i x_i \end{aligned} \tag{12}$$

In Equation (12), $x_i = 1$ if $o_i \in S$ (i.e., if $o_i$ is selected to be put in the knapsack), where $1 \leq i \leq n$. Furthermore, $y_j = 1$ if the total weight of the selected objects in $S$ to be put in the knapsack is $j$, where $1 \leq j \leq W$. The number of QUBO variables is of O($n + W$), which scales linearly with the problem size $n$ plus $W$, and the QUBO formula has both the constraint term and the optimization term. Note that if we consider the number $b = \log W$ of bits representing $W$ as the input size, then the number of QUBO variables is of O($n + 2^b$), which is no longer scales linearly with the problem size. This situation is like that 0/1 KP is regarded as a pseudo polynomial time-complexity problem, as a dynamic programming algorithm can solve the 0/1 KP with the time complexity of O($n \times W$) [45]. However, we still take the value of $W$ as the problem input size and assume that the number of QUBO variables is of O($n + W$), which scales linearly with the problem size $n$ and $W$.

### E. THE QAA SOLVING THE GCP

The graph coloring problem (GCP) is defined as follows:

Given a chromatic number $n$, and an undirected graph $G = (V, E)$ with the vertex set $V$ and the edge set $E$ of $m$ edges, the

GCP is to decide if it is possible to color all vertices in $V$ such that for every edge $(u, v)$ in $E$, vertices $u$ and $v$ have different colors.

The QAA solving the GCP utilizes the following QUBO formula:

$$H(x) = \sum_{v \in V} \left(1 - \sum_{i=1}^{n} x_{v,i}\right)^2 + \sum_{(u,v) \in E} \sum_{i=1}^{n} x_{u,i} x_{v,i} \tag{13}$$

In Equation (13), $x_{v,i} = 1$ indicates that vertex $v$ is colored with color $i$, $1 \leq i \leq n$. The first term and the second term are both constraint terms. The first constraint term means that every vertex should be colored with only one color. The second constraint term means that adjacent vertices should be colored with different colors. The number of QUBO variables is of $O(m \times n)$, which does not scale linearly with the problem size $m$ and $n$, and the QUBO formula has only the constraint term.

### F. THE QAA SOLVING THE HCP

The Hamiltonian cycle problem (HCP) is defined as follows:

Given an undirected graph $G = (V, E)$ with the vertex set $V$ of $n$ vertices denoted by numbers $1, \ldots, n$ and the edge set $E$, the HCP is to decide if there exists a Hamiltonian cycle starting at an arbitrary vertex $s$, visiting very other vertex exactly once, and going back to vertex $s$.

The QAA solving the HCP utilizes the following QUBO formula:

$$H(x) = \sum_{v=1}^{n} \left(1 - \sum_{j=1}^{n} x_{v,j}\right)^2 + \sum_{j=1}^{n} \left(1 - \sum_{v=1}^{n} x_{v,j}\right)^2 + \sum_{(u,v) \notin E} \sum_{j=1}^{n} x_{u,j} x_{v,j+1} \tag{14}$$

In Equation (14), $x_{v,j} = 1$ represents that vertex $v$ is the $j^{th}$ vertex visited, where $1 \leq v, j \leq n$. There are three terms in the QUBO formula. They are all constraint terms. The first term restricts that every vertex can be visited only once. The second term restricts that only one vertex can be visited at a time. The third term restricts that if vertex $v$ is visited after vertex $u$ is visited, then there must be an edge $(u, v) \in E$. The number of QUBO variables is of $O(n^2)$, which does not scale linearly with the problem size $n$, and the QUBO formula has only the constraint term.

### G. THE QAA SOLVING THE TSP

The travelling salesman problem (TSP) is defined as follows:

Given a directed graph $G = (V, E)$ with the vertex set $V$ of $n$ vertices denoted by numbers $1, \ldots, n$ and the edge set $E$, the TSP is to find a cycle that first visits an arbitrary vertex $s$, then sequentially visits every other vertex exactly once, and at last visits vertex $s$, such that the sum of weights of edges included in the cycle is minimized.

The QAA solving the TSP utilizes the following QUBO formula:

$$H(x) = A \sum_{v=1}^{n} \left(1 - \sum_{j=1}^{n} x_{v,j}\right)^2 + A \sum_{j=1}^{n} \left(1 - \sum_{v=1}^{n} x_{v,j}\right)^2 + A \sum_{(u,v) \notin E} \sum_{j=1}^{n} x_{u,j} x_{v,j+1} + B \sum_{(u,v) \in E} W_{u,v} \sum_{j=1}^{n} x_{u,j} x_{v,j+1} \tag{15}$$

In Equation (15), $x_{v,j} = 1$ represents that vertex $v$ is the $j^{th}$ vertex to be visited, where $1 \leq v, j \leq n$. There are four terms in the QUBO formula. The first three terms are constraint terms whose weights are $A$. The first constraint term indicates that each vertex should be visited only once, the second term means that only one vertex should be visited at a time, and the third term means that if the visit of vertex $u$ is followed by the visit of vertex $v$, then there should exist an edge $(u, v) \in E$. The fourth term is an optimization term, in which $W_{u,v}$ is the weight associated with the edge $(u, v)$. The number of QUBO variables is of $O(n^2)$, which does not scale linearly with the problem size $n$, and the QUBO formula has both the constraint term and the optimization term.

### H. THE QAA SOLVING THE JSP

The job shop scheduling problem (JSP) is defined as follows:

Consider a set $\mathcal{M} = \{M_1, \ldots, M_m\}$ of $m$ machines and a set $\mathcal{J} = \{J_1, \ldots, J_n\}$ of $n$ jobs, where job $J_c$ consists of a sequence $JO_c$ of operations for $1 \leq c \leq n$. Let $JO_1 = (o_1, \ldots, o_{k_1}), JO_2 = (o_{k_1+1}, \ldots, o_{k_2}), \ldots, JO_n = (o_{k_{n-1}+1}, \ldots, o_{k_n})$, where $k_n$ is the total number of operations. Note that $k_0$ is set as 0 to be used later. Each operation is associated with an index $i$, $1 \leq i \leq k_n$, and its processing time is denoted as $p_i$. An operation needs to be processed on a specific machine, operations of a job should be processed sequentially, and only one operation of a job can be processed at a given time. The JSP is to find a schedule to assign operations to machines for minimizing the makespan, that is, the total length of the schedule, or the latest finish time of operations.

The QAA solving the JSP utilizes the following QUBO formula:

$$H(x) = A\,h_1(x) + B\,h_2(x) + C\,h_3(x) + D\,h_o(x), \text{ where}$$

$$h_1(x) = \sum_{c=1}^{n}\left(\sum_{\substack{k_{c-1}<i<k_c \\ t+p_i>t'}} x_{i,t}x_{i+1,t'}\right)$$
$$h_2(x) = \sum_{d=1}^{m}\left(\sum_{(i,t,i',t')\in R_d} x_{i,t}x_{i',t'}\right)$$
$$h_3(x) = \sum_{i=0}^{k_n}\left(\sum_{t=1}^{T} x_{i,t} - 1\right)^2$$
$$h_o(x) = \sum_{i=0}^{k_n}\sum_{t=0}^{T}\left(x_{i,t}\,(k_n+1)\right)^{t+p_i} \quad (16)$$

In Equation (16), $x_{i,t} = 1$ represents that operation $o_i$ starts at time $t \le T$, where $T$ is the maximum time. Furthermore, $R_d = A_d \cup B_d, A_d = \{(i,t,i',t') : (i,i') \in I_d \times I_d, i \neq i', 0 \le t, t' \le T, 0 < t'-t < p_i\}, B_d = \{(i,t,i',t') : (i,i') \in I_d \times I_d, i < i', t = t', p_i > 0, p_{i'} > 0\}$, and $I_d$ is the set of indices of all operations that are restricted to be executed on machine $M_d$, $1 \le d \le m$.

The terms $h_1(x), h_2(x)$, and $h_3(x)$ are constraint terms. The term $h_1(x)$ restricts that all operations of a job should be processed sequentially. The term $h_2(x)$ restricts that a machine can process only one operation at a time. The term $h_3(x)$ restricts that every operation should be processed exactly once. The term $h_o(x)$ is the optimization term for minimizing the makespan, since $h_o(y) < h_o(z)$, where $y$ is the optimal solution, and $z$ is any non-optimal (but feasible) solution.

The number of QUBO formula variables is of O($k_n \times T$), which does not scale linearly with the problem size $k_n$ and $T$, and the QUBO formula has both the constraint term and the optimization term.

## IV. THE QUBO FORMULA CLASSIFICATION

As just shown in the last section, QUBO formulas are used by QAAs to solve the SSP, MCP, VCP, 0/1 KP, GCP, HCP, TSP, and JSP. The QUBO formulas can be classified into four classes according to the following two classification criteria.

- Classification criterion 1 (CC1): Does the number of QUBO variables have a linear relationship with the problem input size?

  Some QUBO formulas meet CC1 and maintain a linear relationship between the number of QUBO variables and the problem input size, whereas some QUBO formulas do not. For example, the QUBO formulas used by the QAAs to solve the SSP, MCP, VCP, and 0/1 KP meet CC1. On the contrary, the QUBO formulas used by the QAAs to solve the GCP, HCP, TSP, and JSP do not meet CC1.
- Classification criterion 2 (CC2): Does the QUBO formula have both the constraint term and the optimization term?

  Some QUBO formulas do not meet CC2 and have either the constraint term or the optimization term. However, some QUBO formulas meet CC2 and have both the constraint term and the optimization term. For example, the QUBO formulas used by the QAAs to solve the SSP, MCP, GCP, and HCP have either the constraint term or the optimization term. On the contrary, the QUBO formulas used by the QAAs to solve the VCP, 0/1 KP, TSP, and JSP have both the constraint term and the optimization term.

As shown in Table 2, QUBO formulas can be classified into four classes according to the two criteria CC1 and CC2. Table 2 also shows examples for each class. Specifically, QUBO formulas used by QAAs to solve the SSP and the MCP belong to Class-1. QUBO formulas used by QAAs to solve the VCP and the 0/1 KP belong to Class-2. QUBO formulas used by QAAs to solve the GCP and the HCP belong to Class-3. QUBO formulas used by QAAs to solve the TSP and the JSP belong to Class-4.

## V. BENCHMARKS OF QAAS SOLVING NP-HARD PROBLEMS

This section benchmarks the above-mentioned QAAs against the CAs with the best solutions ever known (i.e., the best quality of the solution). It presents the performance comparisons of the QAAs and related CAs for solving the SSP, MCP, VCP, 0/1 KP, GCP, HCP, TSP, and JSP. The performance comparison experiments are conducted by accessing the D-Wave Advantage quantum annealer through the Amazon Braket service for running the QAAs. The performance information of the CAs solving the same problems is derived from research papers in the literature. Certainly, the QAAs and the CAs are applied to the same problem instances for the sake of fair comparisons. The hardware and software specifications that significantly influence the CA execution time are not the same for different problems, though. We still include the CA execution time derived from existing papers for reference in benchmarking.

### A. BENCHMARKING THE QAA SOLVING THE SSP

The public problem instances, p01, p02, p03, p04, p05, p06, and p07, derived from [46] are used for benchmarking algorithms solving the SSP. Every integer element in set $S$ of the SSP instance is between 5 and 30, and the target integer $T$ is between 20 and 200. The comparative CA is a dynamic programming algorithm [46].

As shown in Table 3, the CA can find correct solutions (indicated by "Y") to all problem instances. However, the QAA cannot find the correct solution to the problem instance p03, in which set $S$ contains many large integers. In other words, the CA can find solutions to all problem instances, so does the QAA except for one problem instance. Thus, the QAA is almost as good as the CA in terms of the quality to solutions, but the CA is a little better. However, the QAA usually consumes less time, either in terms of the total execution time (ET) or the QPU time (QT). For problem instances p02 and p03, the QAA is faster than the CA by a factor around 30, and 130, respectively. Note that below a result in blue with a superscript =, a result in red with a superscript +, and

**TABLE 2.** The four-class classification of QUBO formulas.

| CC1 / CC2 | Yes | No |
|---|---|---|
| No | Class-1<br>(e.g., QUBO formulas to solve the SSP and the MCP) | Class-3<br>(e.g., QUBO formulas to solve the GCP and the HCP) |
| Yes | Class-2<br>(e.g., QUBO formulas to solve the VCP and the 0/1 KP) | Class-4<br>(e.g., QUBO formulas to solve the TSP and the JSP) |

*CC1: Does the number of QUBO variables have a linear relationship with the problem input size?
*CC2: Does the QUBO formula have both the constraint term and the optimization term?

**TABLE 3.** Benchmarks of algorithms solving the SSP.

| SSP Inst. | p01 | p02 | p03 | p04 | p05 | p06 | p07 |
|---|---|---|---|---|---|---|---|
| CA Sol. | Y | Y | Y | Y | Y | Y | Y |
| CA ET | 0.024 | 3.463 | 123.466 | 0.029 | 0.051 | 0.008 | 0.027 |
| QAA Sol. | $Y^{=}$ | $Y^{=}$ | $N^{-}$ | $Y^{=}$ | $Y^{=}$ | $Y^{=}$ | $Y^{=}$ |
| QAA ET | 0.093 | 0.117 | 0.952 | 0.134 | 0.097 | 0.065 | 0.118 |
| QAA QT | 0.077 | 0.031 | 0.323 | 0.066 | 0.057 | 0.051 | 0.081 |

a result in green with a superscript - are used to indicate that the QAAs' solutions (Sol.) are equal to, better than, and worse than those of CAs, respectively. Also note that the time unit is ''second'' in the tables of benchmarks.

## B. BENCHMARKING THE QAA SOLVING THE MCP

The problem instances, g22, g23, g24, g25, g27, g32, g33, g35, g36, and g37, derived from a publicly available database [47] are used for benchmarking algorithms solving the MCP. The instances include cyclic graphs, planar graphs, and random graphs with edge weights of 1, 0, and -1. The number of graph nodes in the instances ranges from 800 to 3000. The CA to be compared is the optimization software provided by Meta-Analytics [48].

As shown in Table 4, both the QA and the CA can find good solutions to the MCP instances, and the QA can even find better solutions to problem instances g33 and g36, improving the maximum cut solution of the weight sum from 1380 to 1382, and from 7674 to 7675, respectively. Due to the limited number of qubits, when faced with large-sized problem instances, it is necessary to first decompose the problem instance by a tool running on a classical computer before embedding it in the QPU for quantum annealing. Therefore, the QAA has longer execution time than the CA, as shown in Table 4. However, the QPU time of the QAA is shorter than the execution time of the CA. Note that EnergyImpactDecomposer, which is a problem decomposition tool provided in the D-Wave Ocean library [49], is used to decompose the problem into smaller subproblems based on the energy contributions of variables to the problem.

## C. BENCHMARKING THE QAA SOLVING THE VCP

The public problem instances, p-hat300-1, keller4, brock400-2, keller5, DSJC500.5, C1000.9, and keller6, derived from the DIMACS (Discrete Mathematics and Theoretical Computer Science) challenge [50] are used for benchmarking algorithms solving the VCP. The number of vertices in vertex set $V$ of the VCP instances is between 300 and 1000. The comparative CA is a branch-and-bound algorithm [51].

There are two QAAs, QAA1 and QAA2, for benchmarking. The QAA1 sets the optimization weight $B$ as 1 and then sets the penalty weight $A$ by employing applicable WSMs in the following order: MOC, MOMC, VLM, MQC, and UB, to make the penalty weight larger and larger until feasible solutions are found. With our experiences, small penalty weights usually lead to infeasible solutions, whereas large penalty weights tend to cause feasible solutions. However, too large penalty weights may lead to feasible solutions with low qualities. This is the reason why QAA1 sets the penalty weight by employing applicable WSMs according to the order of MOC, MOMC, VLM, MQC, and UB. Note that a WSM may not be applicable to a QUBO formula. For example, the MOC cannot be applied to the QUBO formula using matrices of different dimensions.

The penalty weights of the QAA2 are set by running an evolutionary algorithm, the genetic algorithm (GA) provided in [52], for many iterations. The GA encodes penalty weights as bit vectors to go through the population initialization, fitness evaluation, elitism selection, crossover, and mutation processes. The GA details are described as follows. The initial population size is 10. The QAA is used to evaluate the fitness value. The elite number of the roulette-wheel elitism selection process is 4. The crossover rate is 0.9, and the one-point crossover is adopted for the crossover process. The mutation rate is 0.2, and the one-point mutation is employed to flip the bit at the mutation point. The maximum number of iterations to run the GA is 50.

As shown in Table 5, the QAA1 uses the UB method to find solutions to the three problem instances, keller4, keller5, and keller6, whereas it uses the MOC method to find solutions to the other problem instances. QAA1 has equally good solutions as the CA to three problem instances, p-hat300-1, keller4, and DSJC500.5, and has worse solutions than the CA to other problem instances. The QAA2 has equally good solutions to five problem instances, and has worse solutions than the CA to two problem instances, keller 5 and keller 6.

**TABLE 4.** Benchmarks of algorithms solving the MCP.

| MCP Inst. | g22 | g23 | g24 | g25 | g27 | g32 | g33 | g35 | g36 | g39 |
|---|---|---|---|---|---|---|---|---|---|---|
| CA Sol. | 13359 | 13344 | 13337 | 13340 | 3341 | 1410 | 1380 | 7678 | 7674 | 2408 |
| CA ET | 86.5 | 52.1 | 45.7 | 114.6 | 10.4 | 116.6 | 103.3 | 196.4 | 171.6 | 125.4 |
| QAA Sol. | 13359$^{=}$ | 13344$^{=}$ | 13337$^{=}$ | 13340$^{=}$ | 3341$^{=}$ | 1410$^{=}$ | 1382$^{+}$ | 7678$^{=}$ | 7675$^{+}$ | 2408$^{=}$ |
| QAA ET | 1632.5 | 1658.1 | 1799.8 | 1879.6 | 2486.9 | 1450.9 | 1475.8 | 2210.0 | 2070.0 | 3212.6 |
| QAA QT | 14.5 | 16.3 | 11.4 | 17.1 | 34.3 | 29.9 | 30.2 | 17.1 | 17.6 | 34.4 |

**TABLE 5.** Benchmarks of algorithms solving the VCP.

| VCP Inst. | p-hat300-1 | keller4 | brock400-2 | keller5 | DSJC500.5 | C1000.9 | keller6 |
|---|---|---|---|---|---|---|---|
| CA Sol. | 292 | 160 | 371 | 745 | 487 | 932 | 3298 |
| CA ET | 0.560 | 0.006 | 0.935 | 2.380 | 177.790 | 0.498 | 186.540 |
| QAA1 WSM | MOC | UB | MOC | UB | MOC | MOC | UB |
| QAA1 A,B | 1,1 | 171,1 | 1,1 | 776,1 | 1,1 | 1,1 | 3361,1 |
| QAA1 Sol. | 292$^{=}$ | 160$^{=}$ | 375$^{-}$ | 750$^{-}$ | 487$^{=}$ | 933$^{-}$ | 3313$^{-}$ |
| QAA1 ET | 40 | 21.43 | 22.94 | 37.84 | 36.620 | 16.305 | 47.08 |
| QAA1 QT | 0.125 | 0.054 | 0.123 | 0.179 | 0.119 | 0.116 | 0.122 |
| QAA2 WSM | GA | GA | GA | GA | GA | GA | GA |
| QAA2 A,B | 2,1 | 2,1 | 2,1 | 1.5,1 | 2,1 | 1.5,1 | 2,1 |
| QAA2 Sol. | 292$^{=}$ | 160$^{=}$ | 371$^{=}$ | 749$^{-}$ | 487$^{=}$ | 932$^{=}$ | 3305$^{-}$ |
| QAA2 ET | 53 | 37.22 | 34.13 | 49.2 | 47.797 | 20.756 | 171.7 |
| QAA2 QT | 0.126 | 0.117 | 0.121 | 0.213 | 0.178 | 0.107 | 0.268 |

Furthermore, it can be observed that QAAs may be faster or slower than the CAs for the VCP instances.

## D. BENCHMARKING THE QAA SOLVING THE 0/1 KP

The public problem instances, L3, L4, L6, L7, L11, and L14, derived from [47] are used for benchmarking algorithms solving the 0/1 KP. The number of objects of the 0/1 KP instances is between 4 and 45, and the knapsack capacity is between 20 and 1000. The comparative CA is Google OR-Tools [53], which is a free and open-source toolkit, using the linear programming, mixed-integer programming, constraint programming, and vehicle routing algorithms, to solve optimization problems.

There are two QAAs, QAA1 and QAA2, for benchmarking. The QAA1 sets the optimization weight $B$ as 1 and then sets the penalty weight $A$ by employing applicable WSMs in the following order: MOC, MOMC, VLM, MQC, and UB, to make the penalty weight larger and larger until feasible solutions are found. Unfortunately, none of the WSMs can properly set the penalty weights to find feasible solutions. The acronym "NF" representing "Not Found" is used to indicate such cases in Table 6. The penalty weights of the QAA2 are set by running the GA provided in [52]. The GA parameters are set as follows. The number of bits to represent the penalty weight as an integer is 14. The initial population size is 10. The elite number of the selection process is 4. The crossover rate is 0.5, and the one-point crossover is adopted for the crossover process. The mutation rate is 0.2, and the one-point mutation is employed to flip the bit at the mutation point. The maximum number of iterations to run the GA is 100.

As shown in Table 6, the QAA2 and the CA have the same solutions to most problem instances; however, the QAA has worse solutions to problem instances L11 and L14 than the CA has. Furthermore, it can be observed that the CA has better computation time than QAAs.

## E. BENCHMARKING THE QAA SOLVING THE GCP

The public problem instances, R125.1, DSJC125.1, DSJC125.5, R250.1, DSJC250.1, DSJC250.5, DSJC500.1, and le450_15d, derived from [54] are used for benchmarking algorithms solving the GCP. The number of vertices in vertex set $V$ of the GCP instances is between 125 and 450. The comparative CA is a memetic algorithm combining the teaching-learning concept and the tabu-search concept [55].

As shown in Table 7, the QAA cannot find solutions to GCP instances DSJC125.5, DSJC250.5, DSJC500.1, and le450_15d, whereas and the CA can find solutions to all problem instances. Furthermore, the QAA may be faster or slower than the CA for some problem instances.

## F. BENCHMARKING THE QAA SOLVING THE HCP

The public problem instances, 4_H, 6_H-1, 6_H-2, 8_H-1, 8_H-2, and alb1000, derived from [56] are used for benchmarking algorithms solving the HCP. The number of vertices in vertex set $V$ of the HCP instances is between 1000 and 5000. The comparative CA is an algorithm using the "snakes and ladders" heuristic [57]. The algorithm is implemented in Python and C++ [58].

As shown in Table 8, the QAA cannot find solutions to the problem instance alb1000 (indicated by "NF"), whereas and the CA can find solutions to all problem instances. The CA execution time is 0.023 for the alb1000 problem instance. However, the CA execution time cannot be found from research papers and indicated by "N/A" (i.e., "not available") in Table 8. The QAA takes several seconds to run, and takes QPU time less than 0.202 seconds. However, the QAA cannot run properly for the alb1000 instance; it gets an error message of "Kernel died" (indicated by "X" in Table 8) and no solution is returned. The reason for getting such an error message is that the problem size is too large, causing too many QUBO variables.

**TABLE 6.** Benchmarks of algorithms solving the 0/1 KP.

| 0/1 KP Inst. | L3 | L4 | L6 | L7 | L11 | L14 |
|---|---|---|---|---|---|---|
| CA Sol. | 35 | 23 | 52 | 107 | 1437 | 2033 |
| CA ET | 0.0005 | 0.001 | 0.002 | 0.008 | 0.001 | 0.001 |
| QAA1 WSM | UB | UB | UB | UB | UB | UB |
| QAA1 A,B | 48,1 | 41,1 | 105,1 | 188,1 | 1552,1 | 2018,1 |
| QAA1 Sol. | NF | NF | NF | NF | NF | NF |
| QAA1 ET | 19.89 | 12.05 | 81.23 | 101.63 | 115.8 | 82.98 |
| QAA1 QT | 0.426 | 0.316 | 0.360 | 0.689 | 0.476 | 0.398 |
| QAA2 WSM | GA | GA | GA | GA | GA | GA |
| QAA2 A,B | 5, 2000 | 10, 2000 | 3, 2000 | 3, 2000 | 10, 10000 | 5, 2500 |
| QAA2 Sol. | $35^{=}$ | $23^{=}$ | $52^{=}$ | $107^{=}$ | $985^{-}$ | $1679^{-}$ |
| QAA2 ET | 21.86 | 12.75 | 131.93 | 75.65 | 79.4 | 145.06 |
| QAA2 QT | 0.113 | 0.103 | 0.131 | 0.213 | 0.209 | 0.116 |

**TABLE 7.** Benchmarks of algorithms solving the GCP.

| GC Prob. | R125.1 | DSJC125.1 | DSJC125.5 | R250.1 | DSJC250.1 | DSJC250.5 | DSJC500.1 | le450_15d |
|---|---|---|---|---|---|---|---|---|
| CA Sol. | Y | Y | Y | Y | Y | Y | Y | Y |
| CA ET | 0.001 | 0.029 | 0.143 | 0.002 | 0.017 | 10.9 | 720.3 | 983 |
| QAA Sol. | $Y^{=}$ | $Y^{=}$ | $N^{-}$ | $Y^{=}$ | $Y^{=}$ | $N^{-}$ | $N^{-}$ | $N^{-}$ |
| QAA ET | 15.647 | 14.577 | 47.199 | 19.054 | 30.11 | 47.199 | 139.687 | 238.58 |
| QAA QT | 0.117 | 0.118 | 0.184 | 0.116 | 0.184 | 0.194 | 0.31 | 0.38 |

**TABLE 8.** Benchmarks of algorithms solving the HCP.

| GC Prob. | 4_H | 6_H-1 | 6_H-2 | 8_H-1 | 8_H-2 | alb1000 |
|---|---|---|---|---|---|---|
| CA Sol. | Y | Y | Y | Y | Y | Y |
| CA ET | N/A | N/A | N/A | N/A | N/A | 0.023 |
| QAA Sol. | $Y^{=}$ | $Y^{=}$ | $Y^{=}$ | $Y^{=}$ | $Y^{=}$ | NF |
| QA ET | 16.209 | 22.019 | 22.82 | 35.269 | 13.782 | X |
| QAA QT | 0.104 | 0.027 | 0.202 | 0.082 | 0.025 | X |

## G. BENCHMARKING THE QAA SOLVING THE TSP

The public problem instances, br17, ftv33, ftv35, gr17, gr21, p43, ry48p, and kro124p, derived from TSPLIB [59] are used for benchmarking algorithms solving the TSP. The number of vertices of the TSP instances is between 17 and 124. The comparative CA is the algorithm provided by Google OR-Tools [53].

There are three QAAs, QAA1, QAA2, and QAA3, for benchmarking. Again, the QAA1 sets the optimization weight $B$ as 1 and then sets the penalty weight $A$ by employing applicable WSMs in the following order: MOC, MOMC, VLM, MQC, and UB, to make the penalty weight larger and larger until feasible solutions are found. The QAA2 uses a method called "total edge weight adjustment (TEWA)" to set the optimization weight $B$ as 1 and then set the penalty weight $A$ as the value of $\frac{n}{m}\sum_{e\in E} w_e$ (i.e., $A$ is the summation of all edge weights times the number $n$ of vertices and divided by the number $m$ of edges). QAA3 sets the optimization weight $B$ as 1 and uses the GA to set the penalty weight $A$ with the following GA parameters. The number of bits to represent the penalty weight is 16. The initial population size is 4. The elite number of the selection process is 2. The crossover rate is 0.9, and the one-point crossover is adopted for the crossover process. The mutation rate is 0.2, and the one-point mutation is employed to flip the bit at the mutation point. The maximum number of iterations to run the GA is 3.

As shown in Table 9, the QAA1, QAA2, QAA3, and the CA find an equally good solution to the problem instance br17, whereas the QAA1, QAA2, and QAA3 have worse solutions to all other problem instances than the CA. Furthermore, the QAA1, QAA2, and QAA3 are slower than the CA for all problem instances in terms of the total execution time. However, the QPU time of the QAA1, QAA2, and QAA3 is smaller than the execution time of the CA for some problem instances.

## H. BENCHMARKING THE QAA SOLVING THE JSP

The public problem instances, ft02, ft03, ft04, ft06, la03, and ft10, provided by OR-Library [60] are used for benchmarking algorithms solving the JSP. The comparative CA is the algorithm provided by Google OR-Tools [53].

The QUBO formula used by the QAA has three constraint terms and one optimization term, each of which has a different weight. So, none of MOC, MOMC, VLM, MQC, and UB is applicable to tune the penalty weights. Instead, GA is adopted as the WSM to tune the weights $A$, $B$, $C$, and $D$. The GA parameters are set as follows. The number of bits to represent the weights is 5. The initial population size is 10 (for ft02, ft03, ft04, and ft06) or 4 (for la03 and ft10). The elite number of the selection process is 4 (for ft02, ft03, ft04, and ft06) or 2 (for la03 and ft10). The crossover rate is 0.9, and the one-point crossover is adopted for the crossover process. The mutation rate is 0.2, and the one-point mutation is employed

**TABLE 9. Benchmarks of algorithms solving the TSP.**

| TSP | br17 | ftv33 | ftv35 | gr17 | gr21 | p43 | ry48p | kro124p |
|---|---|---|---|---|---|---|---|---|
| **CA Sol.** | 39 | 1355 | 1584 | 2085 | 2707 | 5635 | 14682 | 41232 |
| **CA ET** | 0.027 | 0.194 | 0.166 | 0.055 | 0.06 | 0.15 | 0.261 | 1.666 |
| **QAA1 WSM** | MQC | MQC | MQC | MQC | MQC | MQC | MQC | MQC |
| **QAA1 A,B** | 74,1 | 332,1 | 332,1 | 745,1 | 865,1 | 446,1 | 2782,1 | 4536,1 |
| **QAA1 Sol.** | $39^{=}$ | $1881^{-}$ | $2129^{-}$ | $2134^{-}$ | $2807^{-}$ | $5669^{-}$ | $21803^{-}$ | $85905^{-}$ |
| **QAA1 ET** | 62.9 | 152.55 | 169.94 | 100.88 | 104.43 | 175.96 | 140.05 | 630.91 |
| **QAA1 QT** | 0.394 | 0.706 | 0.843 | 0.611 | 0.738 | 0.659 | 0.402 | 0.589 |
| **QAA2 WSM** | TEWA | TEWA | TEWA | TEWA | TEWA | TEWA | TEWA | TEWA |
| **QAA2 A,B** | 10231,1 | 97063062,1 | 97226954,1 | 4149,1 | 6946,1 | 24935,1 | 10053521,1 | 10189098,1 |
| **QAA2 Sol.** | $39^{=}$ | $2088^{-}$ | $2197^{-}$ | $2123^{-}$ | $2909^{-}$ | $5679^{-}$ | $17883^{-}$ | $105918^{-}$ |
| **QAA2 ET** | 35.3 | 167.98 | 158.11 | 61.0 | 90.3 | 70.6 | 130.7 | 421.1 |
| **QAA2 QT** | 0.121 | 0.694 | 0.692 | 0.213 | 0.198 | 0.243 | 0.244 | 0.476 |
| **QAA3 WSM** | GA | GA | GA | GA | GA | GA | GA | GA |
| **QAA3 A,B** | 27546,1 | 43866,1 | 23699,1 | 39217,1 | 16098,1 | 45128,1 | 59260,1 | 43981,1 |
| **QAA3 Sol.** | $39^{=}$ | $1864^{-}$ | $2361^{-}$ | $2103^{-}$ | $2860^{-}$ | $5691^{-}$ | $22071^{-}$ | $102316^{-}$ |
| **QAA3 ET** | 60.77 | 91.31 | 91.86 | 166.74 | 81.54 | 139.04 | 209.58 | 595.77 |
| **QAA3 QT** | 0.333 | 0.417 | 0.471 | 0.937 | 0.353 | 0.488 | 0.648 | 0.549 |

to flip the bit at the mutation point. The maximum number of iterations to run the GA is 10 (for ft02, ft03, ft04, and ft06) or 3 (for la03 and ft10).

As shown in Table 10, the QAA has the same solutions as the CA to the problem instances ft02, ft03, ft04, and ft06. However, the QAA cannot find any feasible solution (indicated by "NF") to the problem instances la03 and ft10. Furthermore, both the execution time and the QPU time of the QAA are longer than the execution time of the CA.

## VI. OBSERVATIONS FROM QAA BENCHMARKS

By the benchmarks of QAAs and CAs for solving different NP-hard problems, we have the following observations and suggestions.

The QAAs using Class-1 QUBO formulas are likely to have performance that is better than or similar to that of related CAs. This is because the number of QUBO variables of a Class-1 QUBO formula scales linearly with the problem input size. Furthermore, a Class-1 QUBO formula has either the constraint term or the optimization term. Thus, a Class-1 QUBO formula usually has fewer QUBO variables and simpler coupling relationships between variables than formulas of other classes. The graph associated with a Class-1 QUBO formula is thus easier to be embedded into the QPU properly, leading to short computation time and annealing time, along with good solutions.

A Class-2 QUBO formula has both the constraint term and the optimization term, but the number of QUBO variables of a Class-2 QUBO formula scales linearly with the problem input size. Thus, a Class-2 QUBO formula usually has fewer QUBO variables than formulas of other classes. However, it may have more complex coupling relationships. On the contrary, a Class-3 QUBO formula has either the constraint term or the optimization term, leading to simpler coupling relationships. However, the number of QUBO variables of a Class-3 QUBO formula does not scale linearly with the problem input size, which implies there are more QUBO variables. By observing benchmarks of QAAs and related CAs just shown earlier, QAAs using Class-2 or Class-3 QUBO formulas have similar or a little worse performance than related CAs.

A Class-4 QUBO formula has both the constraint term and the optimization term, and the number of QUBO variables of a Class-4 QUBO formula does not scale linearly with the problem input size. Thus, a Class-4 QUBO formula has more QUBO variables and more complex coupling relationships than formulas in other classes. Thus, the graph associated with a Class-4 QUBO formula is hard to be embedded into the QPU properly, leading to long computation time and annealing time, along with unfavorable solutions.

For QAAs using Class-2 or Class-4 QUBO formulas that contain both constraint terms and optimization terms, we need to set the penalty weight and the optimization weight before the quantum annealing process starts. It is suggested to set the optimization weight as 1 and then set the penalty weight by employing applicable WSMs in the following order: MOC, MOMC, VLM, MQC, and UB, to make the penalty weight larger and larger until feasible solutions are found. It is observed that small penalty weights may lead to infeasible solutions, whereas large penalty weights tend to cause feasible solutions. However, too large penalty weights may lead to feasible solutions with low qualities. This is the reason why it is suggested to set the penalty weight by employing applicable WSMs according to the order of MOC, MOMC, VLM, MQC, and UB.

It is also suggested to adopt GA for setting the optimization weight and the penalty term properly. The GA encodes weights as bit vectors to go through the population initialization, fitness evaluation, elitism selection, crossover, and mutation processes. The GA runs iteration by iteration until fitness value converges or the maximum iteration is reached. By the QAA benchmarks shown earlier, the GA is promising to set the penalty weight and the optimization weight properly for QAAs to find good solutions. However, the GA consumes much computation time, which is not included in the QAA benchmarks, to set weights properly. It is suggested to use

**TABLE 10.** **Benchmarks of algorithms solving the JSP.**

| **JSP** | **ft02** | **ft03** | **ft04** | **ft06** | **la03** | **ft10** |
|---|---|---|---|---|---|---|
| **CA Sol.** | 9 | 7 | 10 | 55 | 597 | 930 |
| **CA ET** | 0.139 | 0.011 | 0.011 | 0.084 | 0.221 | 134.8 |
| **QAA WSM** | GA | GA | GA | GA | GA | GA |
| **QAA A,B,C,D** | 2,2,1,1 | 1,1,1,1 | 30,30,15,1 | 3,3,15,1 | 1,1,1,1 | 1,1,1,1 |
| **QAA Sol.** | $9^{=}$ | $7^{=}$ | $10^{=}$ | $55^{=}$ | NF | NF |
| **QAA ET** | 4.789 | 6.252 | 25.071 | 93.147 | 565.74 | 2781.7 |
| **QAA QT** | 0.61 | 0.367 | 0.683 | 1.145 | 1.148 | 0.523 |

the GA to set weights for a problem instance PI for the first time. When the problem instance PI is modified slightly, the same weights are still applied to the modified problem instance. Moreover, the same weights can also be applied to problem instances that are similar to PI. Therefore, the GA computation time can be regarded as just a one-time pre-consumption of time.

## VII. CONCLUSION

In this paper, QUBO formulas are classified into four classes according to two criteria: (i) Does the number of QUBO variables scale linearly with the problem input size? (ii) Does the QUBO formula have both the constraint term and the optimization term? The classification is exemplified by QUBO formulas used by QAAs to solve specific NP-hard problems, including the SSP, MCP, VCP, 0/1 KP, GCP, HCP, TSP, and JSP.

We benchmark the QAAs against related CAs. Observations from QAA benchmarks along with derived suggestions are further given for improving QAA performance. CAs and QAAs using Class-2 and Class-3 QUBO formulas provide solutions of similar qualities to NP-hard problems. QAAs may have longer or shorter execution time than CAs. QAAs using Class-1 (resp., Class-4) QUBO formulas are likely to be better (resp., worse) than CAs in terms of the quality of the solution and the time to the solution. It is believed many QAAs will have superior performance in the future when we go beyond the current NISQ era [32], in which quantum devices have only a moderate number of error-prone qubits.

In the future, we plan to investigate more QAAs using different QUBO formulas to solve more problems to exemplify the QUBO formula classification more thoroughly. In addition to the evolutionary algorithm GA, we also plan to study the possibilities of employing various evolutionary algorithms, such as the particle swarm optimization, ant colony optimization, and tabu search algorithms, to properly set weights of constraint and optimization terms for improving QAA performance. Furthermore, we plan to apply the QUBU formulas to developing algorithms for different computing machines that are similar to the quantum annealer, such as the digital annealer (DA) [61] and the coherent Ising machine (CIM) [62], to see if the DA or the CIM can obtain better solutions than the quantum annealer.

## REFERENCES

[1] F. Arute et al., "Quantum supremacy using a programmable superconducting processor," *Nature*, vol. 574, no. 7779, pp. 505–510, 2019.

[2] M. Bozzo-Rey and R. Loredo, "Introduction to the IBM Q experience and quantum computing," in *Proc. 28th Annu. Int. Conf. Comput. Sci. Softw. Eng.*, 2018, pp. 410–412.

[3] M. AbuGhanem and H. Eleuh, "A quantum state tomography study of Google's Sycamore gate on an IBM's quantum computer," Elsevier, Rochester, NY, US, Tech. Rep. SSRN 4316581.

[4] C. McGeoch and P. Farré, "The D-Wave advantage system: An overview," D-Wave Syst., Burnaby, BC, Canada, Tech. Rep. 14-1049A-A, 2020.

[5] S. Abel, N. Chancellor, and M. Spannowsky, "Quantum computing for quantum tunneling," *Phys. Rev. D, Part. Fields*, vol. 103, no. 1, Jan. 2021, Art. no. 016008.

[6] M. Hernandez and M. Aramon, "Enhancing quantum annealing performance for the molecular similarity problem," *Quantum Inf. Process.*, vol. 16, no. 5, p. 133, May 2017.

[7] A. Perdomo-Ortiz, M. Benedetti, J. Realpe-Gómez, and R. Biswas, "Opportunities and challenges for quantum-assisted machine learning in near-term quantum computers," *Quantum Sci. Technol.*, vol. 3, no. 3, Jun. 2018, Art. no. 030502.

[8] R. Y. Li, R. Di Felice, R. Rohs, and D. A. Lidar, "Quantum annealing versus classical machine learning applied to a simplified computational biology problem," *npj Quantum Inf.*, vol. 4, no. 1, p. 14, Feb. 2018.

[9] E. H. Houssein, Z. Abohashima, M. Elhoseny, and W. M. Mohamed, "Machine learning in the quantum realm: The state-of-the-art, challenges, and future vision," *Expert Syst. Appl.*, vol. 194, May 2022, Art. no. 116512.

[10] M. Wilson, T. Vandal, T. Hogg, and E. G. Rieffel, "Quantum-assisted associative adversarial network: Applying quantum annealing in deep learning," *Quantum Mach. Intell.*, vol. 3, no. 1, pp. 1–14, Jun. 2021.

[11] C. F. Higham and A. Bedford, "Quantum deep learning by sampling neural nets with a quantum annealer," *Sci. Rep.*, vol. 13, no. 1, p. 3939, Mar. 2023.

[12] A. Khoshaman, W. Vinci, B. Denis, E. Andriyash, H. Sadeghi, and M. H. Amin, "Quantum variational autoencoder," *Quantum Sci. Technol.*, vol. 4, no. 1, Sep. 2018, Art. no. 014001.

[13] N. Gao, M. Wilson, T. Vandal, W. Vinci, R. Nemani, and E. Rieffel, "High-dimensional similarity search with quantum-assisted variational autoencoder," in *Proc. 26th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Aug. 2020, pp. 956–964.

[14] A. Perdomo-Ortiz, A. Feldman, A. Ozaeta, S. V. Isakov, Z. Zhu, B. O'Gorman, H. G. Katzgraber, A. Diedrich, H. Neven, J. de Kleer, B. Lackey, and R. Biswas, "Readiness of quantum optimization machines for industrial applications," *Phys. Rev. Appl.*, vol. 12, no. 1, Jul. 2019, Art. no. 014004.

[15] A. Perdomo-Ortiz, J. Fluegemann, S. Narasimhan, R. Biswas, and V. N. Smelyanskiy, "A quantum annealing approach for fault detection and diagnosis of graph-based systems," *Eur. Phys. J. Special Topics*, vol. 224, no. 1, pp. 131–148, Feb. 2015.

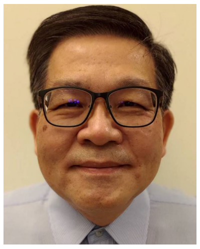
[16] H. Wang, W. Wang, Y. Liu, and B. Alidaee, "Integrating machine learning algorithms with quantum annealing solvers for online fraud detection," *IEEE Access*, vol. 10, pp. 75908–75917, 2022.

[17] J. Lang, S. Zielinski, and S. Feld, "Strategic portfolio optimization using simulated, digital, and quantum annealing," *Appl. Sci.*, vol. 12, no. 23, p. 12288, Dec. 2022.

[18] M. Mattesi, L. Asproni, C. Mattia, S. Tufano, G. Ranieri, D. Caputo, and D. Corbelletto, "Financial portfolio optimization: A QUBO formulation for Sharpe ratio maximization," 2023, *arXiv:2302.12291*.

[19] E. G. Rieffel, D. Venturelli, B. O'Gorman, M. B. Do, E. M. Prystay, and V. N. Smelyanskiy, "A case study in programming a quantum annealer for hard operational planning problems," *Quantum Inf. Process.*, vol. 14, no. 1, pp. 1–36, Jan. 2015.

[20] M. Klar, P. Schworm, X. Wu, M. Glatt, and J. C. Aurich, "Quantum annealing based factory layout planning," *Manuf. Lett.*, vol. 32, pp. 59–62, Apr. 2022.
[21] A. Mott, J. Job, J.-R. Vlimant, D. Lidar, and M. Spiropulu, "Solving a Higgs optimization problem with quantum annealing for machine learning," *Nature*, vol. 550, no. 7676, pp. 375–379, Oct. 2017.
[22] D. Pires, Y. Omar, and J. Seixas, "Adiabatic quantum algorithm for multijet clustering in high energy physics," *Phys. Lett. B*, vol. 843, Aug. 2023, Art. no. 138000.
[23] R. Sandt, Y. Le Bouar, and R. Spatschek, "Quantum annealing for microstructure equilibration with long-range elastic interactions," *Sci. Rep.*, vol. 13, no. 1, p. 6036, Apr. 2023.
[24] K. Wils and B. Chen, "A symbolic approach to discrete structural optimization using quantum annealing," 2023, *arXiv:2307.00153*.
[25] R. Sandt and R. Spatschek, "Efficient low temperature Monte Carlo sampling using quantum annealing," *Sci. Rep.*, vol. 13, no. 1, p. 6754, 2023.
[26] P. Hauke, H. G. Katzgraber, W. Lechner, H. Nishimori, and W. D. Oliver, "Perspectives of quantum annealing: Methods and implementations," *Rep. Prog. Phys.*, vol. 83, no. 5, May 2020, Art. no. 054401.
[27] B. Altshuler, H. Krovi, and J. Roland, "Anderson localization makes adiabatic quantum optimization fail," *Proc. Nat. Acad. Sci. USA*, vol. 107, no. 28, pp. 12446–12450, Jul. 2010.
[28] S. Boixo, T. F. Rønnow, S. V. Isakov, Z. Wang, D. Wecker, D. A. Lidar, J. M. Martinis, and M. Troyer, "Evidence for quantum annealing with more than one hundred qubits," *Nature Phys.*, vol. 10, no. 3, pp. 218–224, Mar. 2014.
[29] T. F. Rønnow, Z. Wang, J. Job, S. Boixo, S. V. Isakov, D. Wecker, J. M. Martinis, D. A. Lidar, and M. Troyer, "Defining and detecting quantum speedup," *Science*, vol. 345, no. 6195, pp. 420–424, Jul. 2014.
[30] H. G. Katzgraber, "Viewing vanilla quantum annealing through spin glasses," *Quantum Sci. Technol.*, vol. 3, no. 3, Jul. 2018, Art. no. 030505.
[31] J.-R. Jiang and C.-W. Chu, "Solving NP-hard problems with quantum annealing," in *Proc. IEEE 4th Eurasia Conf. IoT, Commun. Eng. (ECICE)*, Oct. 2022, pp. 406–411.
[32] J. Preskill, "Quantum computing in the NISQ era and beyond," *Quantum*, vol. 2, p. 79, Aug. 2018.
[33] R. C. T. Lee, R. C. Chang, Y. T. Tsai, and S. S. Tseng, *Introduction to the Design and Analysis of Algorithms*, New York, NY, USA: McGraw-Hill, 2005.
[34] S. A. Cook, "The complexity of theorem-proving procedures," in *Proc. 3rd Annu. ACM Symp. Theory Comput. (STOC)*, 1971, pp. 151–158.
[35] R. M. Karp, "Reducibility among combinatorial problems," in *Complexity of Computer Computations*. Boston, MA, USA: Springer, 1972, pp. 85–103.
[36] S. Yarkoni, E. Raponi, T. Bäck, and S. Schmitt, "Quantum annealing for industry applications: Introduction and review," *Rep. Prog. Phys.*, vol. 85, no. 10, Oct. 2022, Art. no. 104001.
[37] M. Razavy, *Quantum Theory of Tunneling*. Singapore: World Scientific, 2013.
[38] A. Verma and M. Lewis, "Penalty and partitioning techniques to improve performance of QUBO solvers," *Discrete Optim.*, vol. 44, May 2022, Art. no. 100594.
[39] M. Ayodele, "Penalty weights in QUBO formulations: Permutation problems," in *Proc. Eur. Conf. Evol. Comput. Combinat. Optim. (Part EvoStar)*. Cham, Switzerland: Springer, 2022, pp. 159–174.
[40] S. S. Wald, "Thermalisation and relaxation of quantum systems," Ph.D thesis, Dept. Natural Sci. Technol., Université de Lorraine, Lorraine, France, 2017.
[41] A. Lucas, "Ising formulations of many NP problems," *Frontiers Phys.*, vol. 2, p. 5, Feb. 2014.
[42] F. Glover, G. Kochenberger, and Y. Du, "A tutorial on formulating and using QUBO models," 2018, *arXiv:1811.11538*.
[43] G. Bass, M. Henderson, J. Heath, and J. Dulny, "Optimizing the optimizer: Decomposition techniques for quantum annealing," *Quantum Mach. Intell.*, vol. 3, no. 1, pp. 1–14, Jun. 2021.
[44] E. Pelofske, G. Hahn, and H. N. Djidjev, "Solving larger maximum clique problems using parallel quantum annealing," 2022, *arXiv:2205.12165*.
[45] M. R. Garey and D. S. Johnson, *Computers and Intractability*, vol. 174. San Francisco, CA, USA: Freeman, 1979.
[46] *Data for the Subset Sum Problem*. Accessed: Mar. 25, 2023. [Online]. Available: https://people.sc.fsu.edu/~jburkardt/datasets/subset_sum/subset_sum.html
[47] C. Helmberg and F. Rendl, "Solving quadratic (0,1)-problems by semidefinite programs and cutting planes," *Math. Program.*, vol. 82, no. 3, pp. 291–315, Aug. 1998.
[48] *Meta-Analytics: Max Cut Benchmarks*. Accessed: Jun. 1, 2022. [Online]. Available: http://meta-analytics.net/index.php/max-cut-benchmarks/
[49] *D-Wave Decomposer*. Accessed: Mar. 25, 2023. [Online]. Available: https://docs.ocean.dwavesys.com/en/stable/docs_hybrid/reference/decomposers.html,
[50] *Network Repository: A Scientific Network Data Repository With Interactive Visualization and Mining Tools*. Accessed: Mar. 25, 2023. [Online]. Available: https://networkrepository.com/index.php
[51] L. Wang, S. Hu, M. Li, and J. Zhou, "An exact algorithm for minimum vertex cover problem," *Mathematics*, vol. 7, no. 7, p. 603, Jul. 2019.
[52] H. Cheng, *The Genetic Algorithm Solving the Optimization Problem*. Accessed: Mar. 25, 2023. [Online]. Available: https://reurl.cc/6NnnEO
[53] *Google OR-Tools*. Accessed: Mar. 25, 2023. [Online]. Available: https://developers.google.com/optimization
[54] *Graph Coloring Instances*. Accessed: Mar. 25, 2023. [Online]. Available: https://mat.tepper.cmu.edu/COLOR/instances.html#XXDSJ
[55] T. Dokeroglu and E. Sevinc, "Memetic Teaching–Learning-Based optimization algorithms for large graph coloring problems," *Eng. Appl. Artif. Intell.*, vol. 102, Jun. 2021, Art. no. 104282.
[56] M. Meringer, "Fast generation of regular graphs and construction of cages," *J. Graph Theory*, vol. 30, no. 2, pp. 137–146, Feb. 1999.
[57] P. Baniasadi, V. Ejov, J. A. Filar, M. Haythorpe, and S. Rossomakhine, "Deterministic, 'snakes and ladders' heuristic for the Hamiltonian cycle problem," *Math. Program. Comput.*, vol. 6, no. 1, pp. 55–75, 2014.
[58] M. T. Lezana, "A Python implementation of the snakes and ladders for solving the Hamiltonian cycle problem using a graphical interface," Bachelor's Thesis, Dept. Comput. Sci., Univ. Basque Country, Leioa, Spain, 2021.
[59] G. Reinhelt. *TSPLIB: A Library of Sample Instances for the TSP (and Related Problems) From Various Sources and of Various Type*. Accessed: Mar. 25, 2023. [Online]. Available: http://comopt.ifi.uniheidelberg.de/software/TSPLIB95
[60] *OR-Library*. Accessed: Mar. 25, 2023. [Online]. Available: http://people.brunel.ac.uk/~mastjjb/jeb/info.html
[61] O. Šeker, N. Tanoumand, and M. Bodur, "Digital annealer for quadratic unconstrained binary optimization: A comparative performance analysis," *Appl. Soft Comput.*, vol. 127, Sep. 2022, Art. no. 109367.
[62] T. Honjo, T. Sonobe, K. Inaba, T. Inagaki, T. Ikuta, Y. Yamada, T. Kazama, K. Enbutsu, T. Umeki, R. Kasahara, K.-I. Kawarabayashi, and H. Takesue, "100,000-spin coherent Ising machine," *Sci. Adv.*, vol. 7, no. 40, Oct. 2021, Art. no. eabh0952.



**JEHN-RUEY JIANG** (Member, IEEE) received the Ph.D. degree in computer science from National Tsing Hua University, Hsinchu, Taiwan, in 1995. He joined Chung-Yuan Christian University as an Associate Professor, in 1995. Then, he joined Hsuan-Chuang University, in 1998, and became a Full Professor, in 2004. He is currently with the Department of Computer Science and Information Engineering, National Central University, Taoyuan, Taiwan. He also leads the Advanced Computing And Networking (ACAN) Laboratory, which focuses on investigating advanced technologies about computing and networking. His research interests include quantum annealing algorithms, universal quantum algorithms, quantum computing, quantum networking, the Internet of Things, the quantum Internet of Things, machine learning/deep learning, and quantum machine learning/deep learning.

**CHUN-WEI CHU** received the B.S. degree from the Department of Applied Mathematics, National Chung Hsing University, Taichung, Taiwan, in 2020, and the M.S. degree from the Department of Computer Science and Information Engineering, National Central University, Taoyuan, Taiwan, in 2022. His research interests include quantum annealing algorithms and the Internet of Things (IoT).