IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 1 July 2024, accepted 2 September 2024, date of publication 6 September 2024, date of current version 16 September 2024.

*Digital Object Identifier 10.1109/ACCESS.2024.3455436*

RESEARCH ARTICLE

# Benchmarks and Recommendations for Quantum, Digital, and GPU Annealers in Combinatorial Optimization

**JEHN-RUEY JIANG[1], (Member, IEEE), YU-CHEN SHU[2], (Member, IEEE), AND QIAO-YI LIN[1]**
[1]Department of Computer Science and Information Engineering, National Central University, Taoyuan City 32001, Taiwan
[2]Department of Mathematics, National Cheng Kung University, Tainan City 701401, Taiwan

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

This work was supported in part by the National Central University (NCU), in part by the National Science and Technology Council (NSTC) under Grant 111-2115-M-006-019 and Grant 112-2119-M-A49-011, in part by the Center for Quantum Frontiers of Research and Technology (QFort), and in part by the Higher Education Sprout Project, Ministry of Education to the Headquarters of University Advancement at National Cheng Kung University (NCKU).

**ABSTRACT** Annealers leverage quadratic unconstrained binary optimization (QUBO) formulas to address combinatorial optimization problems (COPs) and have shown potential to outperform classical computers. This paper examines three prominent types of annealers: quantum, digital, and GPU annealers. Quantum annealers (QAs) are exemplified by the D-Wave Advantage, which relies on the quantum tunneling phenomenon to rapidly locate the minimum-energy system state corresponding to the optimal solution to a COP. Digital annealers (DAs) are typified by the Fujitsu Digital Annealing Unit (DAU), which is based on a quantum-inspired digital architecture to perform parallel and real-time optimization calculations to solve a COP. GPU annealers (GPUAs) are exemplified by the Compal Quantix solver, which harnesses graphics processing units (GPUs) to conduct the diverse adaptive bulk search for the optimal COP solution. This paper first provides an introductory overview of the QA, DA, and GPUA, and then proceeds to benchmark their performance on solving various well-known COPs such as the subset sum, maximum cut, vertex cover, 0/1 knapsack, graph coloring, Hamiltonian cycle, traveling salesperson, and job-shop scheduling problems. Their performance is also compared with that of state-of-the-art algorithms running on classical computers. Through comprehensive performance benchmarks in terms of the solution quality and the execution time, we identify the strengths and weaknesses of each annealer. In addition, we also provide recommendations to improve the performance of each annealer. Specifically, we recommend using the genetic algorithm, the improved particle swarm optimization algorithm, and the ant colony optimization algorithm in all annealers, using quantum pausing, quenching, and reverse annealing in the QA, using built-in separated penalty terms, one-way/two-way one-hot constraints, and linear inequality constraints in the DA, and implementing some parallel algorithms in the GPUA for performance improvement.

**INDEX TERMS** Combinatorial optimization problem, digital annealer, GPU annealer, quantum annealer, quadratic unconstrained binary optimization.

## I. INTRODUCTION

Quantum computers operate on quantum bits (qubits) and utilize quantum phenomena such as quantum superposition, quantum entanglement, and quantum tunneling for computation. A qubit exists in a superposition of both 0 and 1 simultaneously, collapsing into a definite state of 0 or 1 only upon measurement. In contrast, classical computers operate on bits, where each bit can only be either 0 or 1, but not both simultaneously. Quantum computers have sparked extensive research in recent years due to their ability to provide computational and state representation capabilities

The associate editor coordinating the review of this manuscript and approving it for publication was Sawyer Duane Campbell.

that classical computers cannot, a property known as quantum supremacy [1].

Quantum annealers, such as the D-Wave Advantage system with 5760 qubits organized as the Pegasus architecture [2], are quantum computers that utilize the quantum tunneling phenomenon to address specific problems, such as combinatorial optimization problems (COPs). A COP usually has an objective function with a large solution space in which local optima exist along with the global optimum. Quantum tunneling enables rapid traversal of the energy landscape to locate the global minimum energy corresponding to the optimal COP solution [3]. With the quantum tunneling phenomenon, when a quantum particle encounters an energy barrier, it may tunnel through the barrier even if its kinetic energy is less than the barrier. For a given objective function, a quantum annealer first prepares a state associated with the function. It then initiates an adiabatic process, starting from a quantum superposition state that encompasses all possible candidate states in the solution space with equal probability amplitudes. Subsequently, the amplitudes of all states change simultaneously, akin to traversing all states simultaneously. The adiabatic process controls the probability amplitudes to change slowly enough to halt at a state close to the minimum Hamiltonian (i.e., the minimum total energy) associated with the objective function, which corresponds to the optimal solution of the COP. As illustrated in Figure 1, quantum annealers exploit the quantum tunneling effect to traverse the entire solution space simultaneously, avoiding entrapment in local optima (or local minima) and enabling the identification of the global optimum solution (or global minimum) of the COP.

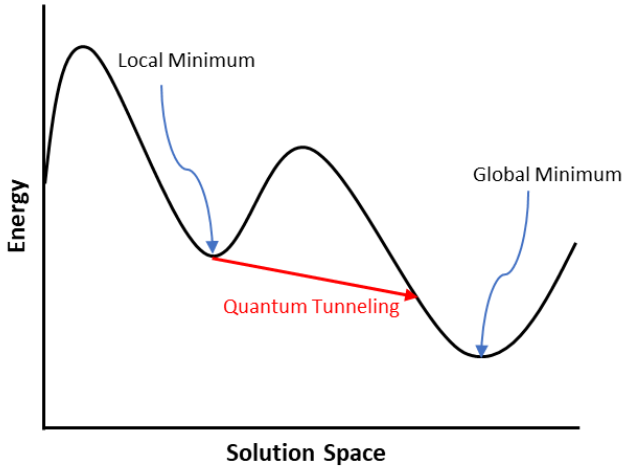


**FIGURE 1. Illustration of the quantum annealing process utilizing the quantum tunneling phenomenon to traverse the energy landscape (solution space) for finding the global minimum in the solution space [3].**

To utilize a quantum annealer to solve a COP, the objective function of the COP must be formulated as a quadratic unconstrained binary optimization (QUBO) formula having linear and quadratic terms of binary variables of 0 and 1. The QUBO formula is then embedded into the architecture of a quantum annealer to undergo the quantum annealing process for many times (shots). The quantum annealer is expected to locate the optimal value of the QUBO formula in a few seconds or tens of seconds. However, the optimal value may not be located due to the following major problems: (i) The number of qubits owned by the quantum annealer is limited. For example, D-Wave 2000Q and Advantage systems have 2048 and 5760 qubits, respectively. (ii) The connectivity of qubits is limited. For example, the maximum qubit connectivity is 6 for the D-Wave 2000Q system, and 15, for the Advantage system. (iii) There exists undesirable noise, such as flux noise [4], to make the quantum annealer fail to locate the QUBO formula's optimal value. Influenced by quantum annealers, quantum-inspired annealers are developed to mitigate the problems of solving complex COPs using QUBO formulas. Typical quantum-inspired annealers include those based on the application-specific integrated circuit (ASIC) [5], field-programmable gate array (FPGA) [6], and graphics processing unit (GPU) [7].

This paper shows comparative analysis for benchmarking three prominent types of annealers, either quantum-based or quantum-inspired, when they are used to solve COPs with QUBO formulas. The annealers for comparisons are the D-Wave Advantage quantum annealer (QA) based on quantum tunneling phenomenon, the Fujitsu digital annealer (DA) based on the ASIC design for parallel real-time optimization, and the Compal GPU-based annealer (GPUA), which harnesses GPUs to perform the diverse adaptive bulk search (DABS) to locate the optimal solution of QUBO formulas. After providing an introductory overview of the three types of annealers, we then benchmark the QA, DA, and GPUA against state-of-the-art classical algorithms (CAs) on solving various well-known COPs, including the subset sum, maximum cut, vertex cover, 0/1 knapsack, graph coloring, Hamiltonian cycle, traveling salesperson, and job-shop scheduling problems. Based on the benchmarking results, we identify the strengths and weaknesses of each annealer, facilitating informed selection for specific optimization tasks. We also recommend possible ways to improve the performance of each annealer.

The rest of this paper is organized as follows. Section II covers background information, including the basic concepts of QUBO, methods to better formulate QUBO formulas, and fundamental introductions to the QA, DA, and GPUA. Performance benchmarking results of the annealers and CAs are shown in Section III. Section IV shows thorough comparisons of the annealers and recommendations to improve the performance of each annealer. Finally, Section V concludes the paper.

## II. PRELIMINARIES

### A. QUBO FUNDAMENTALS

A QUBO formula $f$ of binary variables is expressed as a quadratic equation, as defined in the following Equation (1).

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

where $Q$ represents 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 consisting of $n$ binary variables with values of either 0 or 1. When a COP is modeled as a QUBO formula, the COP can be solved by minimizing the QUBO formula or by deriving the optimal solution to the QUBO formula.

It is noteworthy that $x_i$ is restricted to 0 or 1, allowing us to rewrite Equation (1) as Equation (2) presented below. Additionally, both Equation (1) and Equation (2) hold true for the Ising formula with variables taking values of $-1$ or 1, a model widely known as equivalent to the QUBO formula.

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

Ideally, a QUBO formula should not include any constraint term, as indicated by its name. However, certain COPs come with constraints related to feasible solutions. Any equality constraint $Rx = t$ can be converted 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 referred to as the constraint weight or penalty weight of a real-number value. Integrating the penalty term $\alpha(Rx - t)^2$ with the optimization term (or cost term) $x^T Q x$ extends the QUBO formula $f(x)$ to the following Equation (3).

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

Equation (3) is sometimes generalized into the following Equation (4).

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

where $\beta$ denotes the optimization weight, a real-number value associated with the optimization term $x^T Q x$.

To ensure the entire QUBO formula is minimized and returns an optimal and feasible solution to the formula, it is crucial to appropriately set the values of weights $\alpha$ and $\beta$. Various weight setting methods (WSMs) [8], [9] have been devised for this purpose. To simplify the weight setting, the WSMs focus on setting $\alpha$ by assuming $\beta$ as 1. They consider the following Equation (5) derived from Equation (3):

$$f(x) = \alpha(Rx - t)^2 + x^T Q x = \alpha g(x) + c(x), \tag{5}$$

where $g(x)$ denotes the constraint function, and $c(x)$ represents the cost function or optimization function.

In Equation (5), $g(x) > 0$ indicates an infeasible solution for $x$, while $g(x) = 0$ indicates a feasible solution. Let $y$ be the optimal solution such that $f(y)$ attains the minimal value, and $S$ be the space comprising all infeasible solutions. This leads to the following inequality:

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

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

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

Different WSMs [8], [9] are proposed to set $\alpha$ based on Equation (7). These methods include 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). For detailed insights into setting the penalty weight $\alpha$, readers are referred to [8] and [9].

The penalty weights corresponding to the mentioned WSMs are displayed in Table 1. In this table, $G$ and $C$ represent $n \times n$ matrices that denote $g(x)$ and $c(x)$, respectively. Additionally, $z$ is a column vector filled with all 1's. Thus, $\alpha_{UB}$ serves as an upper bound for the objective function, assuming all QUBO formula coefficients are positive. $\alpha_{MQC}$ represents the maximum QUBO formula coefficient. $\alpha_{VLM}$ provides an estimation of the numerator (i.e., $c(y) - c(x)$) in Equation (7) without accounting for the denominator (i.e., $g(x)$). $\alpha_{MOMC}$ enhances $\alpha_{VLM}$ by considering $g(x)$ and estimating it as $\gamma$, which is the minimum change in the constraint function exceeding 0. Finally, $\alpha_{MOC}$ aims to further refine $\alpha_{VLM}$ by contemplating a potential increase in the constraint function due to a modification in the objective function, achieved by toggling any bit from 0 to 1 or vice versa. For detailed insights into setting the penalty weight $\alpha$, readers are referred to [3], [8], [9].

### B. QUANTUM ANNEALER BASICS

D-Wave has designed and built a series of QAs, including the 2000Q released in 2017 and the Advantage released in 2020. These systems are based on the quantum tunneling phenomenon to solve COPs with QUBO formulas. In this subsection, we use the D-Wave Advantage system as an example to introduce the basics of quantum annealers.

The D-Wave Advantage system is designed to have a quantum processing unit (QPU) of 5760 qubits organized in the Pegasus architecture [2], as shown in Figure 2. The architecture's topology is not fully connected; rather, it is sparse, with each qubit connected to at most 15 other qubits. Two qubits are linked via a coupler that provides various strengths, enabling fine-tuned control over the interaction between qubits. The Advantage system has a total of 40484 couplers.

Figure 3 illustrates the workflow for a QA, such as the D-Wave Advantage system, to solve a COP. Five major steps of the workflow are further elaborated below.

Step 1 Problem Formulation: The COP is first modeled as a QUBO formula. This formula is translated into a graph structure, where nodes represent binary variables, and edge weights symbolize the coupling strength between variables.

Step 2. Minor Embedding: The graph derived from the QUBO formula is embedded into the QPU. Qubits and couplers map to graph nodes and edges, respectively. Due to hardware limitations, direct connections are restricted, leading to the use of multiple qubits of strong couplers to

**TABLE 1. Different weight setting methods (WSMs) and their associated penalty weights [3].**

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

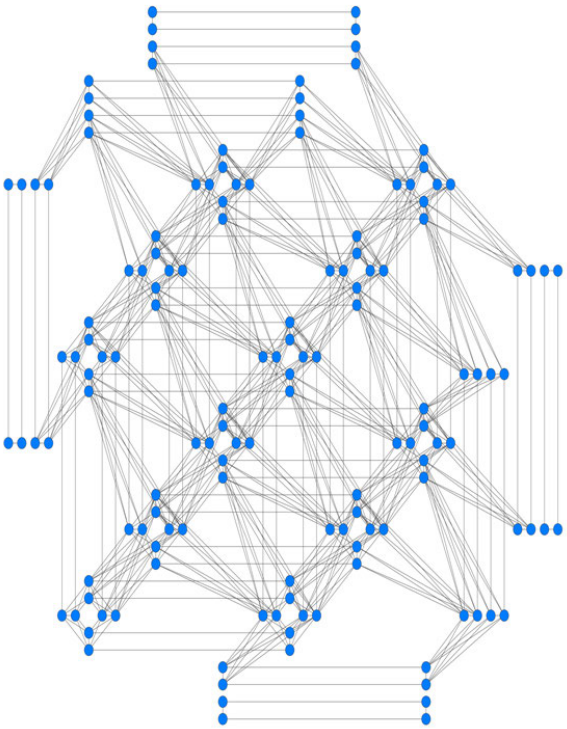

**FIGURE 2. Illustration of the pegasus topology of the D-Wave advantage quantum processing unit [10].**

logically represent a node. The strong couplers are specially called chains, as exemplified by a double line between two nodes in Figure 3. Qubits connected by chains are expected to have the same value; however, chains may sometimes be broken to make qubits have different values. In such cases, mechanisms like majority voting are employed to determine a consensus value.

When the number of qubits required surpasses the QPU maximum capacity, the graph corresponding to the original COP must be decomposed into subgraphs to be properly embedded into the QPU. Various methods for graph decomposition are known, including the iterative centrality halo approach [11], which focuses on nodes with significant impacts on the overall solution, and the DBK (Decomposition, Bounds, K-core) method [12], which recursively breaks down a graph into subgraphs of predefined sizes. The choice of graph decomposition method has a profound impact on the performance of quantum annealers solving COPs.

Step 3. This step defines the initial Hamiltonian $H_i$ and the final Hamiltonian $H_f$ for the entire system. Each Hamiltonian represents the total energy of the system in a specific state. $H_i$ is configured to maintain qubits in a superposition state, while $H_f$ is tailored based on the QUBO formula to ensure that the minimum final Hamiltonian corresponds to the optimal solution to the COP at hand. The system's Hamiltonian $H(t)$ at time $t$ during quantum annealing is expressed in 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 over the annealing time $t$. $A(t)$ transitions gradually from 1 to near 0, while $B(t)$ transitions from 0 to near 1.

Step 4. Annealing: This step performs an annealing process to make the system have the minimum Hamiltonian. The system begins with the lowest initial Hamiltonian, where each qubit is in a superposition state. Throughout annealing, the initial Hamiltonian decreases gradually, while the final Hamiltonian increases gradually. Eventually, the impact of the initial Hamiltonian diminishes to zero, and the system settles into the lowest energy state of the final Hamiltonian associated with the QUBO formula. This transition causes each qubit to collapse from superposition to either 0 or 1, representing the binary variable value that achieves the globally minimum (or optimal) QUBO formula value.

The energy scaling functions $A(t)$ and $B(t)$ in Equation (8), depicted in the lower right part of Figure 3, illustrate changes during the annealing process. As time $t$ progresses, $A(t)$ gradually decreases, while $B(t)$ increases. Consequently, the influence of the final Hamiltonian $H_f$ becomes more pronounced, while the impact of the initial Hamiltonian $H_i$

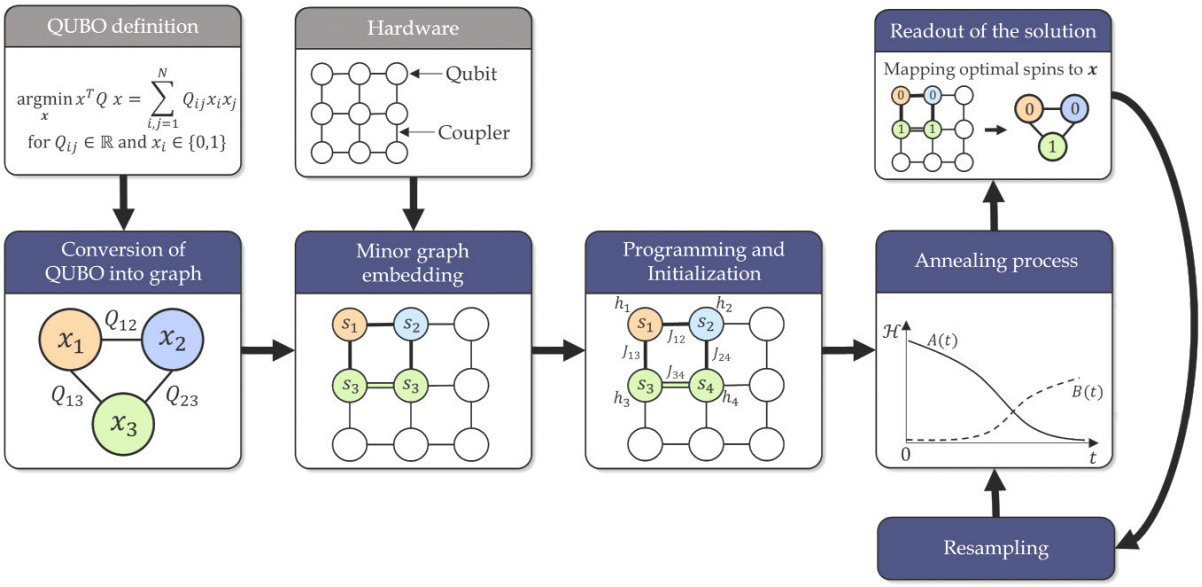


**FIGURE 3.** The workflow for the D-Wave advantage QA to solve a COP with a QUBO formula [3].

decreases. By the end of annealing, the behavior of the system aligns mainly with the final Hamiltonian $H_f$.

Step 5. Reading: Upon completing the annealing process, each qubit's value (0 or 1) is retrieved for subsequent processing. Utilizing the relationship between the qubit and the graph node, a solution to the initial COP can be deduced. In cases where qubits representing the same node exhibit discrepancies due to broken chains, post-processing techniques like majority voting are utilized to ascertain the node's value. It's important to note that Steps 4 and 5 are iterated multiple times (shots), known as the resampling process as shown in the lowest right part of Figure 3, to enhance the probability of discovering the optimal solution to the problem.

QAs are likely to have good performance in solving intricate COPs. As mentioned in [3], they have already applied to solving COPs arising in different application areas, such as quantum chemistry [13], quantum machine learning [14], [15], [16], quantum deep learning [17], [18], quantum variational autoencoders [19], [20], fault detection and diagnosis [21], [22], online fraud detection [23], financial portfolio optimization [24], [25], operational planning [26], [27], data processing in high energy physics [28], [29], material microstructure equilibration [30], [31] and Monte Carlo sampling [32].

### C. DIGITAL ANNEALER BASICS

The digital annealer (DA) developed by Fujitsu Ltd. is a new technology inspired by QAs. It is a hardware implementation (CMOS-based ASIC) that operates at room temperature and uses digital circuitry for performing the simulated annealing (SA) algorithm to solve complex COPs. The SA algorithm is a Markov chain Monte Carlo (MCMC) method for approximating the global optimum of the cost function of a COP. It is inspired by the annealing process in metallurgy, where a metal is heated and then slowly cooled to achieve its lowest-energy state to have good properties of ductility, toughness, plasticity, and hardness.

The SA algorithm has a special "temperature variable" $T$. It selects an initial solution, and sets a high "temperature" value to $T$ and decreases it iteratively. It then randomly perturbs the current solution to generate a new solution. The difference ($\Delta E$) in energy values (i.e., cost function values) between the current and new solutions is calculated. If $\Delta E < 0$, the new solution is better, and it is always accepted. If $\Delta E > 0$, the new solution is worse, and it is accepted with a probability given by the Boltzmann distribution: $P(acceptance) = e^{\frac{-\Delta E}{T}}$. Based on such probability distribution, a worse solution may also be accepted, and the probability of accepting a worse solution drops with decreasing temperatures. The SA decreases $T$ according to a cooling schedule and continues to generate new solutions iteration by iteration until a stopping criterion is met, such as a maximum number of iterations or a minimum temperature value.

The DA uses parallel-trial schemes, such as one-bit inversion, and starts all trials from the same solution to save the calculations of the initial energy value and the energy value differences. Figure 4 shows the one-bit inversion mechanism used by the DA to find the optimal cost function value quickly by inversion of one bit in parallel [33]. The DA also develops an escaping mechanism called a dynamic offset to increase the acceptance probability when no bit-inversion proposal is accepted. In addition, the DA uses parallel tempering with iso-energetic cluster moves to enhance the exploration capabilities [34], [35]. In summary, the DA

can assess numerous options simultaneously, delivering fast insights for large-scale COPs that were once unsolvable, enhancing productivity and efficiency without the need for classical brute-force methods. All the features above makes that the DA has the superior ability to solve large and complex COPs in QUBO formulation, such as the quadratic assignment problem (QAP), quadratic knapsack problem (QKP), quadratic cycle partition problem (QCPP), selective graph coloring problem (SGCP), max cut problem (MCP), vertex cover problem (VCP), and warehouse assignment problem (WAP) with remarkable speed and accuracy [35], [36], [37], [38]. Various practical applications have also shown the capacity of DAs, for example, middle-molecule drug discovery and delivery planning [36].

In this study, we focus on the third-generation DA. The problem scale is up to 100000 bits, which is significantly larger than the 8192 bits of the second generation DA. Additionally, the bits are fully connected with coupling coefficients having the precision of a 64-bit signed integer. The DA also separates cost and penalty terms, which allows the DA to more effectively balance objectives and constraints automatically, leading to more efficient and better optimization. It also introduces special functionalities for handling one-way/two-way one-hot constraints and linear inequality constraints directly [36]. These features are convenient to the users who want to solve QUBO formulas with constraints.

Since more constraints are involved in solving QUBO problems, recently, the fourth generation DA leverages constraints to narrow the search space and directly evaluates inequality violations with the help of GPUs during the optimization process. With the assistance of GPUs, the fourth generation DA is supposed to show superior performance to DAs of previous generations and comparable performance to state-of-the-art solvers [39].

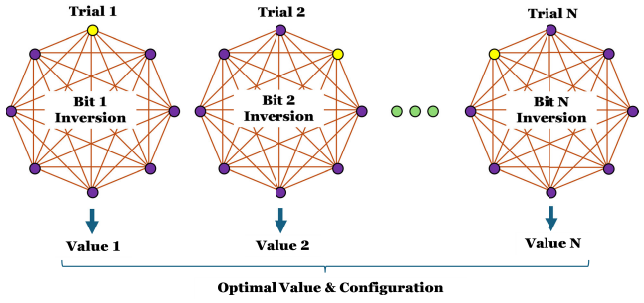


**FIGURE 4. Illustration of the 1-bit inversion mechanism of the Fujitsu DA (adapted from [33]).**

### D. GPU ANNEALER BASICS

The Compal company built a quantum-inspired annealer called Quantix. Since Quantix harnesses the computational power of GPUs to accelerate finding optimal COP solutions, it will be called the GPU annealer (GPUA) in this paper. It has been applied to solve some complex COPs, such as 5G network architecture optimization, energy usage optimization, logistics transportation optimization, and so on.

To our knowledge, Quantix relies on a concept similar to the diverse adaptive bulk search (DABS) proposed in [7] to find the optimal solution to a COP. It is built with 20 Intel Xeon CPUs and 4 NVIDIA GV100 GPUs, each of which has 5120 Compute Unified Device Architecture (CUDA) cores.

The DABS is proposed in [7], of which the framework is shown in Figure 5. As shown in Figure 5, the DABS framework is based on CPUs and GPUs. It has multiple solution pools maintained by a host run on CPUs. Each solution pool is associated with a GPU and contains entries to keep the information of good solutions returned by the GPU. An entry in the solution pool contains a solution (or solution vector), and its associated energy, along with the search algorithm and the genetic operation the derive the solution.

Based on Open Multi-Processing (OpenMP) threads, the host CPUs perform genetic algorithm operations (GAOs) on selected solutions from a solution pool or multiple solution pools to generate so-called target solutions to be sent to GPUs. Based on the received target solutions, local search algorithms (LSAs) are executed by GPUs, each with multiple CUDA blocks of numerous CUDA threads that operate concurrently.

The DABS aims to increase the diversity of GAOs and LSAs. It offers five distinct LSAs, namely MaxMin, CyclicMin, RandomMin, PositiveMin, and TwoNeighbor. Each LSA iteratively flips bits in a solution to explore the solution space for improving solutions. During DABS execution, LSAs that yield better solutions are prioritized for more frequent execution. The DABS also offers eight different GAOs, including Mutation, Crossover, Xrossover, Zero, One, IntervalZero, Best, and Random. Similar to LSAs, the GAOs leading to better solutions are executed more frequently than others.

## III. PERFORMANCE BENCHMARKING

This Section benchmarks the above-mentioned annealers against state-of-the-art classical algorithms (CAs). Specifically, it presents the performance comparisons of the QA, DA, GPUA, and related CAs on solving various well-known COPs, including 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 problem (JSP). The QUBO formulas for solving the COPs are also described in this section.

The annealers used for benchmarking are described as follows. D-Wave Advantage released in 2020 is taken as the QA. It has 5640 qubits with 15-way connectivity. Fujutsu DAU-3 released in 2021 is taken as the DA. It is based on a ASIC chip and can simulate the quantum annealing of 100000 fully connected qubits. Compal Quantix released in

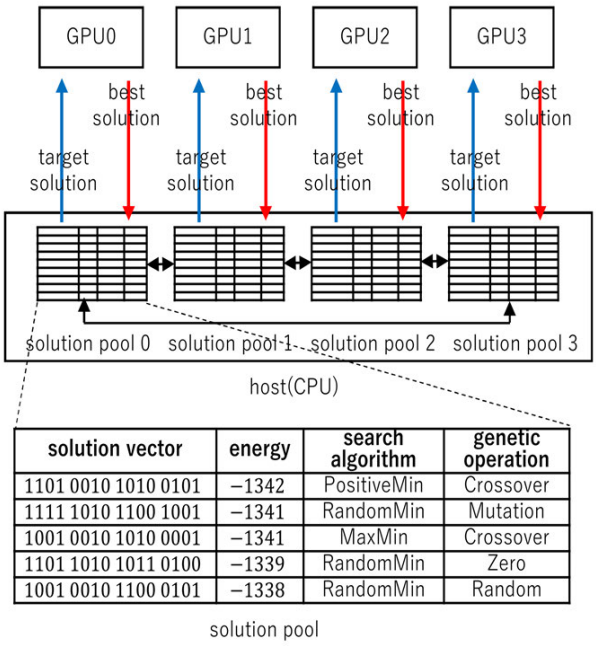


**FIGURE 5.** Illustration of the DABS framework [7] adopted by the Compal Quantix GPUA.

2023 is taken as the GPUA. It is built atop a system with 20 Xeon 2.4-GHz CPUs and four NVIDIA GV100 TITAN-V GPUs, which can simulate the quantum annealing of about 60000 fully connected qubits.

In the following subsections, the COPs used for benchmarking and the best CAs for solving them are introduced. It is notable that special functions provided by each individual annealer to enhance annealing performance are not considered in the benchmarking for fair comparisons. That is to say, all annealers just use the same naive QUBO formulas and suitable WSMs to solve the COPs. The only exception is that we allow the QA to use a library tool for problem decomposition; otherwise, many problem instances cannot be solved on the QA.

The benchmarking results are shown in tables in separate subsections. To enhance visualization, the tables use cells in distinct colors to represent performance ranks: pink for the top rank, brown for the second rank, green for the third rank, and blue for the fourth rank. Moreover, the acronym "IS" in the tables stands for "infeasible solution", and the mark "X" corresponds to a certain error, such as "no matrix is produced due to insufficient memory", and "kernel die", etc. Also note that the results of the best CAs are derived from the literature, whereas the results of the annealers are collected from our own practical experiments.

### A. BENCHMARKS ON THE SSP

The definition of the subset sum problem (SSP) is 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 QUBO formula for solving the SSP is as follows:

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

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 $1 \le i \le n$.

The public problem instances, p01, p02, p03, p04, p05, p06, and p07, derived from [40] are used for benchmarking annealers and a CA solving the SSP. Every integer element in the 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 [40].

As shown in Table 2, the CA, DA, and GPUA can find correct solutions (indicated by "Yes") to all problem instances. However, the QA cannot find the correct solution to the problem instance p03, in which set $S$ contains many large integers like 1648264, 1955584, and 2074132. The reason for the QA problem-solving failure may be due to the insufficient precision of the coupling coefficient. We will further discuss this in the next section.

In terms of the execution time of the annealers, the QA generally has the shortest, the GPUA has the second shortest, and the DA has the longest execution time. The execution time of the CA sometimes is the shortest (e.g., when solving the p01 problem instance), but sometimes it is the longest (e.g., when solving the p03 problem instance). The reasons for the CA to have the longest problem-solving time are as follows. The CA is a dynamic programming algorithm whose execution time is proportional to the given target integer $T$ in the SSP. The p03 problem instance indeed has a very large target integer $T = 2463098$, so the execution time of the CA is very long.

### B. BENCHMARKS ON THE MCP

The definition of the maximum cut problem (MCP) is 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 QUBO formula for solving the MCP is as follows.

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

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 problem instances, g22, g23, g24, g25, g27, g32, g33, g35, g36, and g39, derived from a publicly available database [41] are used for benchmarking annealers and a CA solving the MCP. The instances include cyclic graphs, planar graphs, and random graphs with edge weights of 1, 0, and −1.

**TABLE 2.** Benchmarks of the CA, QA, DA, and GPUA solving the subset sum problem.

| Datasets | p01 | p02 | p03 | p04 | p05 | p06 | p07 |
|---|---|---|---|---|---|---|---|
| CA Sol. | Yes | Yes | Yes | Yes | Yes | Yes | Yes |
| QA Sol. | Yes | Yes | No | Yes | Yes | Yes | Yes |
| DA Sol. | Yes | Yes | Yes | Yes | Yes | Yes | Yes |
| GPUA Sol. | Yes | Yes | Yes | Yes | Yes | Yes | Yes |
| CA Time | 0.024 | 3.463 | 123.466 | 0.029 | 0.051 | 0.008 | 0.027 |
| QA Time | 0.093 | 0.117 | 0.952 | 0.134 | 0.097 | 0.065 | 0.118 |
| DA Time | 6.518 | 6.504 | 6.504 | 6.526 | 6.537 | 6.48 | 6.497 |
| GPUA Time | 1.56 | 1.8223 | 101.09 | 1.735 | 1.77 | 1.496 | 1.78 |

The number of graph vertices in the instances are 2000 with the number of edges ranging from 2000 to 19990. The CA to be compared is the optimization software provided by Meta-Analytics [42].

As shown in Table 3, the DA always jointly possesses the best solutions; it has the highest frequency of jointly possessing the best solutions. The QA has the second highest frequency of jointly possessing the best solutions, the CA has the third highest frequency of jointly possessing the best solutions, and the GPUA has the lowest frequency of jointly possessing the best solutions.

In terms of the execution time, generally speaking, the DA has the shortest execution time, the GPUA has the second shortest, the CA has the third shortest, and the QA has the longest execution time. The reasons for the QA's longest execution time are explained as follows. Due to the limited number of qubits, large problem instances must first be decomposed with a tool running on a classical computer before being embedded in the QPU for quantum annealing. This results in the QA having a longer execution time compared to the others. Notably, the EnergyImpactDecomposer, a problem decomposition tool available in the D-Wave Ocean library [43], is used to break down the problem into smaller subproblems based on the energy contributions of variables.

### C. BENCHMARKS ON THE VCP

The definition of the vertex cover problem (VCP) is 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 QUBO formula for solving the VCP is as follows:

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

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 $B$.

As shown in Table 4, the UB WSM and the MOC WSM are used to set the penalty weight $A$, while the optimization weight $B$ is set to 1. As stated in [3], WSMs in the order of MOC, MOMC, VLM, MQC, and UB progressively increase the penalty weight. Small penalty weights usually lead to infeasible solutions, whereas large penalty weights tend to produce feasible solutions. However, excessively large penalty weights can result in feasible solutions of low quality (i.e., high cost function values). Therefore, the penalty weight is set with an applicable WSM according to the order of MOC, MOMC, VLM, MQC, and UB. Note that not all WSMs can be applied to every QUBO formula. For instance, the MOC cannot be applied to the QUBO formula that involves matrices of varying dimensions.

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 [44] are used for benchmarking annealers and a CA 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 [45].

As shown in Table 4, generally speaking, the CA provides the best solutions, and the QA and the DA offer the second-best solutions, while the GPUA provides worse solutions. The GPUA even returns an "infeasible solution (IS)" for the keller6 problem instance. The reason for the "IS" case may be due to the the large size of the keller6 problem instance, which corresponds to a graph containing about 3400 nodes, 4600000 edges, with the maximum degree of around 3000, and the minimum degree of around 2700 [44].

In terms of the execution time of feasible solutions, generally speaking, the DA has the shortest execution time, followed by the GPUA with the second shortest, the QA with the third shortest, and the CA with the longest execution time. Note that the GPUA's execution time for the infeasible solution (IS) is as long as 355 seconds. This is because we set the maximum execution time of the GPUA as 355 seconds to increase the probability of finding better solutions. However, no feasible solution is found even after the GPUA executes for 355 seconds.

### D. BENCHMARKS ON THE 0/1 KP

The definition of the 0/1 knapsack problem (0/1 KP) is 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

**TABLE 3.** Benchmarks of the CA, QA, DA, and GPUA solving the max cut problem.

| Datasets | g22 | g23 | g24 | g25 | g27 | g32 | g33 | g35 | g36 | g39 |
|---|---|---|---|---|---|---|---|---|---|---|
| CA Sol. | 13359 | 13344 | 13337 | 13340 | 3341 | 1410 | 1380 | 7678 | 7674 | 2408 |
| QA Sol. | 13359 | 13344 | 13337 | 13340 | 3341 | 1410 | 1382 | 7678 | 7675 | 2408 |
| DA Sol. | 13359 | 13344 | 13337 | 13340 | 3341 | 1410 | 1382 | 7686 | 7680 | 2408 |
| GPUA Sol. | 13359 | 13344 | 13336 | 13339 | 3341 | 1406 | 1378 | 7678 | 7676 | 2404 |
| CA Time | 86.5 | 52.1 | 45.7 | 114.6 | 10.4 | 116.6 | 103.3 | 196.4 | 171.6 | 125.4 |
| QA Time | 1632.5 | 1658.1 | 1799.8 | 1879.6 | 2486.9 | 1450.9 | 1475.8 | 2210 | 2070 | 3212.6 |
| DA Time | 10.381 | 10.394 | 10.212 | 10.246 | 10.273 | 10.273 | 10.089 | 30.413 | 30.455 | 10.314 |
| GPUA Time | 22.479 | 12.22 | 12.39 | 22.232 | 12.35 | 12.37 | 102.638 | 12.35 | 102.638 | 12.39 |

**TABLE 4.** Benchmarks of the CA, QA, DA, and GPUA solving the vertex cover problem.

| Datasets | p-hat-300-1 | keller4 | brock400-2 | keller5 | DSJC500.5 | C1000.9 | keller6 |
|---|---|---|---|---|---|---|---|
| CA Sol. | 292 | 160 | 371 | 745 | 487 | 932 | 3298 |
| WSM | MOC | UB | MOC | UB | MOC | MOC | UB |
| A,B | 1,1 | 171,1 | 1,1 | 776,1 | 1,1 | 1,1 | 3361,1 |
| QA Sol. | 292 | 160 | 375 | 750 | 487 | 933 | 3313 |
| DA Sol. | 292 | 160 | 371 | 749 | 487 | 932 | 3302 |
| GPUA Sol. | 292 | 160 | 371 | 749 | 487 | 933 | IS |
| CA Time | 0.56 | 0.006 | 0.935 | 2.38 | 177.79 | 0.498 | 186.54 |
| QA Time | 40 | 21.43 | 22.94 | 37.84 | 36.62 | 16.305 | 47.08 |
| DA Time | 6.575 | 6.5 | 6.567 | 6.634 | 6.591 | 6.838 | 6.969 |
| GPUA Time | 11.37 | 6.196 | 11.498 | 11.544 | 6.607 | 21.879 | 355 |

$\sum_{o_i \in S} w_i \leq W$ (i.e., selected objects can be accommodated by the knapsack and have the maximum total cost).

The QUBO formula for solving the 0/1 KP is as follows:

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

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 public problem instances, L3, L4, L6, L7, L11, and L14, derived from [46] are used for benchmarking annealers and a CA 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 [47], 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.

Similarly, the optimization weight $B$ is set as 1 and the penalty weight $A$ is set 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 for the QA to find feasible solutions. Thus, the UB method is adopted and the acronym "IS" stands to "infeasible solution" for such cases in Table 5.

The reason for the "IS" cases just mentioned may be due to the insufficient number of qubits owned by the QA. Notably, the QUBO formula for solving the 0/1 KP has both the constraint term and the optimization term. Furthermore, it has a number of QUBO variables of $O(n + W)$, which scales with the problem size $n$ plus $W$. Moreover, every pair of variables $x_i$ and $y_j$ has an in-between coupler for $1 \leq i \leq n$ and $1 \leq j \leq W$. Note that $n$ and $W$ may be 45 and as large as 1000, respectively. The QA thus needs to perform node chaining and meanwhile decompose the original 0/1 KP into many subproblems to embed the QUBO formula into the QPU. This makes the setting of the optimization weight and the penalty weight very difficult, making it harder for the QA to return feasible solutions.

Conversely, the DA and the GPUA are less likely to have "IS" cases because they can simulate 100000 and 60000 fully connected qubits, respectively. Surprisingly, the GPUA can find feasible solutions for all problem instances, while the DA fails to find a feasible solution for the L14 problem instance, which has $n = 45$ objects with the knapsack capacity $W = 907$.

As shown in Table 5, generally speaking, the CA has the best solution, the GPUA has the second-best solution, the DA has the third-best solution, whereas the QA has the worst solution and fails to find any feasible solution to the QUBO formulas with penalty weights set by the UB method.

In terms of the execution time, the CA has the shortest execution time, while the DA sometimes has the second shortest execution time, and the GPUA has the second shortest execution time at other times. The QA often has the longest execution time.

**TABLE 5. Benchmarks of the CA, QA, DA, and GPUA solving the 0/1 knapsack problem.**

| Datasets | L3 | L4 | L6 | L7 | L11 | L14 |
|---|---|---|---|---|---|---|
| CA Sol. | 35 | 23 | 52 | 107 | 1437 | 2033 |
| WSM | UB | UB | UB | UB | UB | UB |
| A,B | 48,1 | 41,1 | 105,1 | 188,1 | 1552,1 | 2018, 1 |
| QA Sol. | IS | IS | IS | IS | IS | IS |
| DA Sol. | 35 | 23 | 52 | 107 | 1391 | IS |
| GPUA Sol. | 35 | 23 | 52 | 107 | 1437 | 1431 |
| CA Time | 0.0005 | 0.001 | 0.002 | 0.008 | 0.001 | 0.001 |
| QA Time | 19.89 | 12.05 | 81.23 | 101.63 | 115.8 | 82.98 |
| DA Time | 6.488 | 6.53 | 6.54 | 6.49 | 102.215 | 101.039 |
| GPUA Time | 6.152 | 6.145 | 6.169 | 6.163 | 102.587 | 104.126 |

### E. BENCHMARKS ON THE GCP

The definition of the graph coloring problem (GCP) is 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 QUBO formula for solving the VCP is as follows:

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

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 $O(m \times n)$.

The public problem instances, R125.1, DSJC125.1, DSJC125.5, R250.1, DSJC250.1, DSJC250.5, DSJC500.1, and le450_15d, derived from [48] are used for benchmarking annealers and a CA 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 [49].

As shown in Table 6, both the CA and the DA provide the best solutions; they can find optimal (i.e., "Yes") solutions for all problem instances. The QA has worse solutions than the CA and the DA; it cannot find optimal solutions for four problem instances, namely DSJC125.5, DSJC250.5, DSJC500.1, and le450_15d. The GPUA has the worst solutions; it cannot find optimal solutions for five problem instances, namely DSJC125.5, DSJC250.1, DSJC250.5, DSJC500.1, and le450_15d. The five problem instances are related to a 125-node and 3891-edge graph, a 250-node and 3217-edge graph, a 250-node and 15668-edge graph, a 500-node and 12456-edge graph, and a 450-node and 16680-edge graph, respectively. The five problem instances are all related to graphs having more than 3000 edges, whereas other problem instances are related to graphs having 1473 or less edges. The problem instances having large numbers of edges may be the reason for the QA and the GPUA to fail to have optimal solutions.

In terms of the execution time, generally speaking, the CA has the shortest execution time, whereas the DA has the second shortest execution time. However, for the DSJC500.1 and le450_15d problem instances, the DA has the shortest execution time. Both the QA and the GPUA have longer execution time than the CA and the DA. The GPUA has shorter execution time than the QA for problem instances for which the GPUA and the QA can provide optimal solutions. In contrast, GPUA's execution time is longer than QA's for problem instances where the QA or the GPUA cannot provide the optimal solution, with one exception: problem instance le450_15d. In this instance, GPUA's execution time is shorter than QA's.

### F. BENCHMARKS ON THE HCP

The definition of the Hamiltonian cycle problem (HCP) is 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 QUBO formula for solving the HCP is as follows:

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

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.

The public problem instances, 4_H, 6_H-1, 6_H-2, 8_H-1, 8_H-2, and alb1000, derived from [50] are used for benchmarking annealers and a CA solving the HCP. The number of vertices of the HCP instances is 10, 14, 14, 30, 30, and 1000, respectively, whereas the number of edges is 15, 21, 21, 45, 45, and 1500. The comparative CA is an algorithm using the "snakes and ladders" heuristic [51]. The algorithm is implemented in Python and C++ [52].

**TABLE 6.** Benchmarks of the CA, QA, DA, and GPUA solving the graph coloring problem.

| Datasets | R125.1 | DSJC125.1 | DSJC125.5 | R250.1 | DSJC250.1 | DSJC250.5 | DSJC500.1 | le450_15d |
|---|---|---|---|---|---|---|---|---|
| CA Sol. | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes |
| QA Sol. | Yes | Yes | No | Yes | Yes | No | No | No |
| DA Sol. | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes |
| GPUA Sol. | Yes | Yes | No | Yes | No | No | No | No |
| CA Time | 0.001 | 0.029 | 0.143 | 0.002 | 0.017 | 10.9 | 720.3 | 983 |
| QA Time | 15.647 | 14.577 | 47.199 | 19.054 | 30.11 | 47.199 | 139.687 | 238.58 |
| DA Time | 6.727 | 6.652 | 6.034 | 5.052 | 5.083 | 30.709 | 31.396 | 8.455 |
| GPUA Time | 11.443 | 11.497 | 102.94 | 12.422 | 202.5 | 213.75 | 207.981 | 210.181 |

**TABLE 7.** Benchmarks of the CA, QA, DA, and GPUA solving the hamiltonian cycle problem.

| Datasets | 4-H | 6_H-1 | 6_H-2 | 8_H-1 | 8_H-2 | alb1000 |
|---|---|---|---|---|---|---|
| CA Sol. | Yes | Yes | Yes | Yes | Yes | Yes |
| QA Sol. | Yes | Yes | Yes | Yes | Yes | X |
| DA Sol. | Yes | Yes | Yes | Yes | Yes | X |
| GPUA Sol. | Yes | Yes | Yes | Yes | Yes | X |
| CA Time | N/A | N/A | N/A | N/A | N/A | 0.023 |
| QA Time | 16.209 | 22.019 | 22.82 | 35.269 | 13.782 | – |
| DA Time | 6.643 | 6.508 | 6.507 | 6.514 | 6.539 | – |
| GPUA Time | 6.04 | 6.05 | 6.04 | 6.14 | 6.04 | – |

As shown in Table 7, the CA can find "Yes" solutions to all problem instances. The QA, DA, and GPUA cannot find the "Yes" solution to the alb1000 problem instance, which is related to a graph of 1000 vertices and 1500 edges. Yet, they can find "Yes" solutions to the other problem instances, which are related to graphs of tens of vertices and edges. The solutions of the QA, DA, and GPUA are marked as "X" to indicate they cannot handle the alb1000 problem instance and do not return any solution. For example, the QA just shows an error message of "kernel died" and returns no solution when solving the alb1000 problem instance. The reason for getting such an error message is that the problem size of the alb1000 problem instance is too large, causing too many QUBO variables and couplers. Similarly, the DA and the GPUA cannot handle the alb1000 problem instance, either.

There are some not-available cases in the CA's execution times. The CA's execution time is 0.023 second for the alb1000 problem instance. However, the CA's execution time for other problem instances cannot be found from research papers and is indicated by "N/A" (i.e., "not available") in Table 7. For the 4_H, 6_H-1, 6_H-2, 8_H-1, and 8_H-2 problem instances, the GPUA has the shortest execution times of 6 seconds or more, the DA has the second-shortest execution times of 6.5 seconds or more, and the QA has the longest execution times of tens of seconds. Since the CA's execution time for the large-sized alb1000 problem instance is only 0.023 second, it is reasonable to infer that the CA has shorter execution time than annealers for small-sized 4_H, 6_H-1, 6_H-2, 8_H-1, and 8_H-2 problem instances. Note that the execution times of annealers corresponding to the "X" solutions are marked as "–", which will be ranked as the longest execution times in benchmarking.

### G. BENCHMARKS ON THE TSP

The definition of the travelling salesperson problem (TSP) is 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 QUBO formula for solving the TSP is as follows:

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

In Equation (15), $x_{v,j} = 1$ represents that vertex $v$ is the $j^{th}$ vertex to be visited, where $1 \le v, j \le n$. 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.

The public problem instances, br17, ftv33, ftv35, gr17, gr21, p43, ry48p, and kro124p, derived from TSPLIB [53] are used for benchmarking annealers and a CA solving the TSP. The number of vertices of the br17, ftv33, ftv35, gr17, gr21, p43, ry48p, and kro124p problem instances is 17, 34, 36, 17, 21, 43, 48, and 100, respectively. The edge weights are all integers with the maximum values of 9999, 100000000, 100000000, 745, 775, 5126, 9999999, and

9999999, respectively. The comparative CA is the algorithm provided by Google OR-Tools [47].

The optimization weight $B$ in the QUBO formula is set as 1, and the penalty weight $A$ is set 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. As indicated in Table 8, the MQC mechanism is employed to set the penalty weight $A$, while the optimization weight $B$ is set as 1.

As shown in Table 8, the CA has the best solution to all problem instances except for the ftv35 problem instance, to which the DA has the best solution. The DA and the GPUA provide comparably good solutions, and each has its own strengths. The DA has one "IS" case for the problem instances p43, whereas the GPUA has two IS cases for the problem instances p43 and kro124p. The QA has worse solution than the DA and the GPUA for most problem instances. It also has one "IS" case for the problem instances p43.

In terms of the execution time, the CA has the shortest execution time in the range between 0.027 and 1.666 seconds for all problem instances. Among annealers, the DA has the shortest execution times except for the problem instance p43. The DA's execution time for the p43 problem instance is 61.722 seconds, which is due to our approach: we set the maximum execution time for the DA to be approximately 60 seconds, attempting to find a better solution but ultimately failing, corresponding to an "IS" case. For other problem instances, the DA's execution times are all less than 10 seconds, ranging from 5.974 to 9.765 seconds. For problem instances br17, ftv33, ftv35, gr17, and gr21, the GPUA's execution times are also short, ranging from 6.358 to 12.85 seconds. GPUA's execution time for the problem instance br17 is even shorter than the DA's. However, the GPUA's execution times are around 100 or 200 seconds for problem instances p43, ry48p, and kro124p. The long execution times of the GPUA are again due to our approach: we set the maximum execution times for the GPUA to be approximately 100 or 200 seconds, attempting to find better solutions, resulting in two "IS" cases and one feasible solution, though. The QA generally has long execution times, approximately tens of seconds to a little over one hundred seconds. For the smaller-sized problem instances br17, ftv33, ftv35, gr17, and gr21, the QA's execution times are much longer than the DA's and the GPUA's.

### H. BENCHMARKS ON THE JSP

The definition of the job-shop scheduling problem (JSP) is 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 $1 \le c \le n$. 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 $i$, $1 \le i \le k_n$, 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 QUBO formula for solving the JSP is as follows:

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

where

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

In Equation (16), $x_{i,t} = 1$ represents that operation $o_i$ starts at time $t \le T$, where $T$ is the possible maximum time span. Furthermore, $R_d = A_d \cup B_d, A_d = \{(i,t,i',t') : (i,i') \in I_d \times I_d, i \ne i', 0 \le t, t' \le T, 0 < t' - t < p_i\}, B_d = \{(i,t,i',t') : (i,i') \in I_d \times I_d, i < i', t = t', p_i > 0, p_{i'} > 0\}$, 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. The QUBO formula has three constraint terms and one optimization term, each of which is intended to have a different weight. So, none of MOC, MOMC, VLM, MQC, and UB is applicable to tune the penalty weights in the benchmarking.

The public problem instances, ft02, ft03, ft04, ft06, la03, and ft10, provided by OR-Library [54] are used for benchmarking annealers and a CA solving the JSP. The comparative CA is the algorithm provided by Google OR-Tools [47]. The ft02, ft03, ft04, ft06, la03, and ft10 problem instances have 2, 3, 4, 6, 10, and 10 jobs,

**TABLE 8.** Benchmarks of the CA, QA, DA, and GPUA solving the travelling salesperson problem.

| Datasets | br17 | ftv33 | ftv35 | gr17 | gr21 | p43 | ry48p | kro124p |
|---|---|---|---|---|---|---|---|---|
| CA Sol. | 39 | 1355 | 1584 | 2085 | 2707 | 5635 | 14682 | 41232 |
| WSM | MQC | MQC | MQC | MQC | MQC | MQC | MQC | MQC |
| A,B | 74,1 | 332,1 | 332,1 | 745,1 | 865,1 | 446,1 | 2782,1 | 4536,1 |
| QA Sol. | 39 | 1467 | 1773 | 2099 | 2807 | IS | 17669 | 105691 |
| DA Sol. | 39 | 1392 | 1554 | 2085 | 2707 | IS | 16242 | 48714 |
| GPUA Sol. | 39 | 1382 | 1590 | 2085 | 2707 | IS | 15538 | IS |
| CA Time | 0.027 | 0.194 | 0.166 | 0.055 | 0.06 | 0.15 | 0.261 | 1.666 |
| QA Time | 62.9 | 26.18872 | 30.9965 | 48.3118 | 104.43 | 48.8453 | 85.21 | 135.32695 |
| DA Time | 6.558 | 7.042 | 7.075 | 6.519 | 6.568 | 61.722 | 5.974 | 9.765 |
| GPUA Time | 6.358 | 7.523 | 12.85 | 6.3668 | 6.644 | 104.102 | 104.469 | 216.905 |

respectively. They have 5, 5, 5, 6, 5, and 10 machines, respectively. Their possible maximum time spans are 22, 15, 16, 55, 597, and 930, respectively.

As shown in Table 9, the CA returns the best solution to every problem instance. The three annealers even cannot handle the la03 and ft10 problem instances, which are indicated by "X" in Table 9. This is because these two problem instances have relatively larger time spans (597 and 930, respectively), which necessitates a greater number of QUBO variables and couplers. The three annealers return the same best solutions to the problem instances ft02 and ft03. The DA and the GPUA return the same best solution to the problem instances ft04. Yet, the QA's solution is the worst. For the problem instance ft06, the CA's solution is the best and much better than the annealers'. Among annealers, the QA has the best, the DA has the second-best, and the GPUA has worst solution to the problem instance ft06.

In terms of the execution time, the CA has the shortest execution times. Among annealers, the DA and the GPUA have similar execution times, ranging from 5 to 16 seconds, whereas the QA has much longer execution times, ranging from 20 to 66 seconds. Note that the execution times of annealers are indicated by "–" for the "X" cases. They will be ranked as the longest execution times in benchmarking.

### I. OVERALL BENCHMARK RANKING

The overall solution quality ranking of the CAs and annealers is shown in Table 10, and the ranking distribution is shown in Figure 6. We can observe that the CA has been ranked as the top for a total count of 54 times, which is the highest count. This could be attributed to the CA's development spanning over several decades, incorporating various strategies and techniques to enhance the quality of solutions to COPs. In contrast, for the annealers, we merely utilize the most basic QUBO formulas to solve COPs. We can also observe that the DA has been ranked as the top for a total count of 46, which is the second-highest count; the GPUA has been ranked as the top for a total count of 33, which is the third-highest count; the QA has been ranked as the top for a total count of 31, which is the least frequent. That is, among the three annealers, the DA has the most instances of being ranked as the top. This could be due to the large number (i.e., 100000) of fully connected simulated qubits supported by the DA.

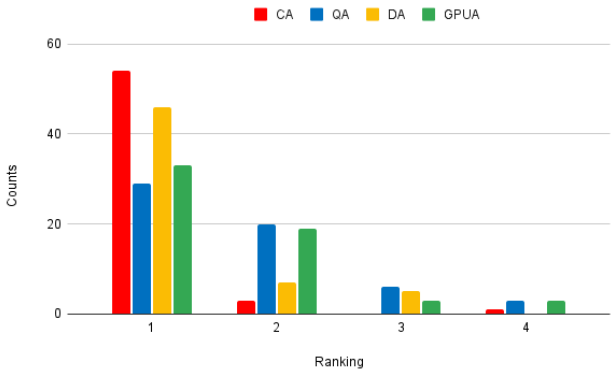


**FIGURE 6.** Solution quality ranking distribution of the CA, QA, DA, and GPUA.

The overall execution time ranking of the CA and annealers is shown in Table 11, and the ranking distribution is shown in Figure 7. The shorter the execution time, the higher the ranking. The execution times corresponding to the "X" cases are listed as "–", and these execution times are ranked the lowest, as the annealers cannot produce any solution in such cases. Additionally, the "N/A" access times of the CAs are ranked the highest, since they are inferred to be the shortest execution times, as mentioned earlier. We can observe that the CA has been ranked as the top for a total count of 42 times, which is the highest count. This means that CAs generally have the shortest execution time. This may be due to the fact that, over the past few decades of development, CA has various methods to reduce the execution time, while annealers have many cases they cannot handle (hence their execution times are ranked the lowest). The DA has been ranked as the top for a total count of 13, which is the second-highest count; the QA has been ranked as the top for a total count of 2, which is the third-highest count; the GPUA has been ranked as the top for a total count of 1, which is the least frequent. That is, among the three annealers, DA has the most instances of being ranked as the top, whereas the QA and the GPUA have very few instances of being ranked as the top. The short execution times of the DA could be due to the DA's using

**TABLE 9.** Benchmarks of the CA, QA, DA, and GPUA solving the job-shop scheduling problem.

| Datasets | ft02 | ft03 | ft04 | ft06 | la03 | ft10 |
|---|---|---|---|---|---|---|
| CA Sol. | 9 | 7 | 10 | 55 | 597 | 930 |
| QA Sol. | 9 | 7 | 12 | 176 | X | X |
| DA Sol. | 9 | 7 | 10 | 179 | X | X |
| GPUA Sol. | 9 | 7 | 10 | 197 | X | X |
| CA Time | 0.139 | 0.011 | 0.011 | 0.084 | 0.221 | 134.8 |
| QA Time | 26.902 | 20.8402 | 32.7974 | 65.9319 | – | – |
| DA Time | 6.502 | 6.512 | 6.511 | 7.284 | – | – |
| GPUA Time | 16 | 5 | 5 | 5 | – | – |

**TABLE 10.** Solution quality ranking counts of the CA, QA, DA, and GPUA.

| Ranking | $1^{st}$ | $2^{nd}$ | $3^{rd}$ | $4^{th}$ |
|---|---|---|---|---|
| CA Sol. | 54 | 3 | 0 | 1 |
| QA Sol. | 29 | 20 | 6 | 3 |
| DA Sol. | 46 | 7 | 5 | 0 |
| GPUA Sol. | 33 | 19 | 3 | 3 |

**TABLE 11.** Execution time ranking counts of the CA, QA, DA, and GPUA.

| Ranking | $1^{st}$ | $2^{nd}$ | $3^{rd}$ | $4^{th}$ |
|---|---|---|---|---|
| CA Time | 42 | 1 | 11 | 4 |
| QA TIme | 2 | 12 | 8 | 36 |
| DA Time | 13 | 22 | 17 | 6 |
| GPUA Time | 1 | 29 | 19 | 9 |

ASIC chips to simulate the annealing process in a parallel manner.

## IV. ANNEALER COMPARISONS AND IMPROVEMENT RECOMMENDATIONS

In this section, we show comparisons of the QA, DA, and GPUA. Furthermore, we give performance improvement recommendations for each annealer.

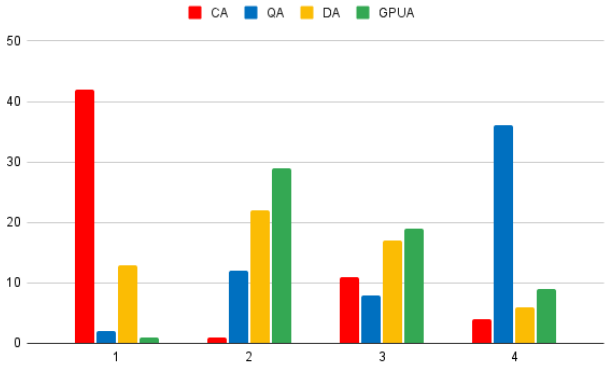


**FIGURE 7.** Execution time ranking distribution of the CA, QA, DA, and GPUA.

### A. ANNEALER COMPARISONS

Table 12 shows comparisons for the QA, DA, and GPUA in terms of the technology used, number of (qu)bits supported, connectivity of (qu)bits, solution quality top rank counts, and execution time top rank counts. The annealers to be compared are as follows. D-Wave Advantage released in 2020 is taken as the QA. Fujutsu DAU-3 released in 2021 is taken as the DA. Compal Quantix released in 2023 is taken as the GPUA.

We can observe from the comparison results that the DA has the highest count of the top rank among annealers, the GPUA has the second highest, and the QA has the third highest. This may be because the DA can support the most qubits, the GPUA supports the second most, and the QA supports the third most. The DA's and the GPUA's qubits are emulated to be fully connected. The QA's qubits are practical; however, they are at most 15-way connected. The QA is a practical quantum annealer. However, the DA and the GPUA are quantum-inspired annealers, they both use LSAs. Supported by the ASIC hardware design, the DA's LSAs (e.g., 1-bit inversion) may have broader search ranges and higher degrees of parallelism than the GPUA's (e.g., MaxMin). This may cause the DA to find better solutions than the GPUA.

### B. QA IMPROVEMENT RECOMMENDATIONS

As shown in Table 2, the QA cannot find the correct solution to the SSP instance p03, which involves many large integers such as 1648264, 1955584, and 2074132. The reason for the QA's problem-solving failure may be because the QA has insufficient precision of the coupling coefficient. The paper [55] proposes a method, called high precision enhancement (HPE) algorithm, to extend the D-Wave 2X to support higher precision coefficients. As mentioned in [55], the D-Wave 2X only guarantees support for coefficients with 4 to 5 bits of resolution or precision. However, we do not have exact information about the coefficient precision of the D-wave Advantage. We believe the precision is about 7 to 8 bits, which is insufficient to differentiate integers ranging from 0 to as large as 2074132, the largest integer of the SSP instance. It is suggested to apply the HPE algorithm to solve the problem.

In [3], the genetic algorithm (GA) is applied to setting the penalty weights and the optimization weights to improve the QA performance for some problems like the VCP, 0/1 KP, TSP, and JSP. The GA can indeed improve the QA performance. That is, the weights set by the GA can enable the QA to have better performance than the weights set by naive WSMs for several problem instances. However, the iteration-based GA invokes the QA at each iteration and thus consumes significant computation time to set weights properly. As stated in [3], it is suggested to use the GA to set weights for a certain problem instance PI for the first time. When PI is modified slightly, the same weights are still applied to the modified PI. Moreover, the same weights can also be applied to other problem instances that are similar to PI. Therefore, the GA computation time can be regarded as just a one-time preprocessing expense. Notably, the GA can also be applied to DA and GPUA to improve their performance. Furthermore, advanced meta-heuristic algorithms similar to the GA, such as the improved particle swarm optimization (PSO) algorithm and the ant colony optimization (ACO) algorithm, are

**TABLE 12. Annealer comparisons.**

| Annealers | QA | DA | GPUA |
|---|---|---|---|
| Device | D-Wave Advantage (2020) | Fujitsu DAU-3 (2021) | Compal Quantix (2023) |
| Technology | Quantum Annealing | ASIC Digital Annealing | GPU-based Annealing |
| Number of (qu)bits | 5,640 | 100,000 | 60,000 |
| Connectivity | At most 15 ways | Fully-connected | Fully-connected |
| Solution quality top rank count | 29 | 46 | 33 |
| Execution time top rank count | 2 | 13 | 1 |

also recommended for enhancing the performance of all annealers.

Besides the (HPE) algorithm and the GA, there are many mechanisms to improve the QA performance, such as pausing [56], [57], [58], quenching [59], [60], and reverse-annealing [61], [62]. Below, we describe every mechanism in detail.

The pausing mechanism involves halting the evolution of the quantum annealing process at a specific point for a specific period of time. It allows the system to equilibrate and potentially find a lower energy state before continuing. Therefore, the likelihood of the system settling into a global minimum rather than a local minimum is increased, enhancing overall performance. The paper [57] focuses on parameter setting for pausing, whereas the paper [58] explores using machine learning technique to identify appropriate pausing locations for better performance.

Quenching in quantum annealing refers to the process of abruptly stopping the quantum annealing process before it reaches its minimum energy state. That is, quenching involves interrupting this quantum annealing process prematurely. Quenching allows faster solution finding, as it bypasses the final slow convergence to the ground state. In some cases, quenching can also help the system escape from local minima trapping the system in a low-energy state that is close to the global minima. The paper [59] investigates the quenching mechanism from an energetic perspective. It analyzes how the energy distribution of the system evolves with different quench rates and how the distribution affects the probability of finding good solutions. The paper [60] investigates quenching in the context of the ECP. It analyzes the relationship between the ''quantum annealing gap'' (the energy difference between the ground state and the excited state) and the success rate of finding optimal solutions with quenching.

Reverse annealing starts from a known solution or a set of candidate solutions and then partially anneals back towards the initial Hamiltonian before resuming forward annealing. This technique allows the system to refine and improve upon known good solutions rather than starting from a random or poorly defined state. For example, we can use the greedy algorithm to easily find a feasible and good (but not optimal) solution to the 0/1 KP. We then apply the reverse annealing mechanism to start from the feasible solution, aiming to ultimately obtain the optimal solution.

### C. DA IMPROVEMENT RECOMMENDATIONS

In the DA (i.e., Fujitsu DAU-3), numerous mechanisms exist to enhance the performance, including built-in separated penalty terms, one-way/two-way one-hot constraints, and linear inequality constraints. Below we demonstrate how these mechanisms are used in the DA for solving COPs.

We first take the 0/1 KP as an example. The built-in ''linear inequality constraints'' feature which allows multiple inequalities in the DA can be used in the 0/1 KP to ensure that the sum of weights of objects in the knapsack does not exceed the knapsack capacity. Therefore, we can remove the term $A\left(\sum_{j=1}^{W} jy_j - \sum_{i=1}^{n} w_i x_i\right)^2$ from the QUBO formula in Eq. (12). Moreover, the built-in ''one-way one-hot group'' feature can be used to ensure that there is only a '1' value among the variables in each group of variables. In this way, we can remove the term $A\left(1 - \sum_{j}^{W} y_j\right)^2$ from the QUBO formula in Eq. (12). In this way, the QUBO formula is reduced significantly and its optimal (or near optimal) value can be derived easily.

We then take the TSP as another example. In the TSP, we may set the one-way one-hot constraint for each vertex in the graph so that each vertex is visited exactly once and only one vertex is visited at a time. We may use the ''two-way one-hot group'' feature provided by the DA such that there is only a '1' value among the variables in each group of variables. In this way, we can remove the terms $A\sum_{v=1}^{n}\left(1 - \sum_{j=1}^{n} x_{v,j}\right)^2$ and $A\sum_{j=1}^{n}\left(1 - \sum_{v=1}^{n} x_{v,j}\right)^2$ from the QUBO formula in Eq. (15). Furthermore, we can use the ''separated penalty terms'' for the DA to adjust the weights $A$ and $B$ in Eq. (15) automatically to soon return the optimal (or near optimal) value of the QUBO formula.

The aforementioned examples show that the QUBO formulation can be simplified by the DA's built-in features. The features not only reduce the burden on users to develop QUBO formulas, but also accelerates the time for the DA to return solutions and improves the quality of the solutions.

### D. GPUA IMPROVEMENT RECOMMENDATIONS

The GPUA is still evolving, and many features have not yet been fully developed. The GPUA is a software annealer which highly depends on parallel algorithms run on CPUs and GPUs. Therefore, more available features, including those

owned by the DA, might be implemented and included in the GPUA to improve the performance.

For example, we can implement some parallel algorithms for the GPUA to handle the inequalities in parallel. Besides, We can specify one-hot constraint in the first $n$ elements with random numbers smaller than $n$.

For another example, in the GPUA, we can create a $n \times n$ matrix where each row contains exactly one '1', and so does each column by using a special permutation of $n$ elements to be the ''one-in-one-out'' feature, which is useful to solve the TSP.

In addition, the GPU has become more and more powerful and popular in the high performance computing. We can expect that one day, with enough GPUs, GPUA may outperform other types of annealers.

## V. CONCLUSION

In this paper, we begin by offering introductory descriptions of the QA, DA, and GPUA to help readers understand their distinct principles, features, and problem-solving workflows. We then conduct comprehensive benchmark analysis of their solution quality and execution time by applying them to solving various COPs, including the SSP, MCP, VCP, 0/1 KP, GCP, HCP, TSP, and JSP. The benchmarking also includes comparisons with the state-of-the-art CAs that solve the same COPs.

Because the COPs selected for benchmarking are all quite well known, after decades of development, CAs for solving the COPs often have very good or even the optimal solutions with very short execution times. In contrast, annealers have only been developed for a few years or for a little over a decade. However, they have been already able to solve COPs systematically via QUBO formulation with relatively good performance or even better performance when compared to CAs. Therefore, when facing new problems that are never or seldom addressed by CAs, using annealers to solve them is very promising.

We also provide performance improvement recommendations for the QA, DA, and GPUA. Specifically, we suggest to use the GA, the improved PSO algorithm, and the ACO algorithm for all annealers to improve solution quality. We suggest to use the HPE algorithm for the QA to support higher precision coefficients, use pausing, quenching, and/or reverse-annealing mechanisms for the QA to improve performance. We are currently in the noisy intermediate-scale quantum (NISQ) era, so the QA only has a moderate number of qubits and is susceptible to noise, which causes its performance to be sometimes inferior to that of the DA and the GPUA. When the hardware specifications of the QA improve, such as having more qubits with higher connectivity and better noise tolerance, as well as supporting coefficients with higher precision (which means finer control over the couplers between qubits), combining hardware improvement with the suggested mechanisms mentioned above may make the QA perform as well as, or even better than, the DA and the GPUA.

We also suggest to use built-in separated penalty terms, one-way/two-way constraints, and linear inequality constraints for the DA to improve performance. Furthermore, we suggest to use parallel inequality checking, and sweeping pattern constraints for the GPUA to improve performance. We believe these recommendations are useful not only for annealer users, but also for annealer developers. We plan to follow these recommendations in the future to verify their actual impact on annealer performance. Last but not least, since energy consumption is also an important metric, we will attempt to incorporate it into future performance benchmarking.

## ACKNOWLEDGMENT

The authors would like to express their sincere gratitude to Compal Electronics, Inc., for providing them with free access to the Quantix service.

## REFERENCES

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

[2] C. McGeoch and P. Farré, *The D-Wave Advantage System: An Overview*. Burnaby, BC, Canada: D-Wave Systems Inc, 2020.

[3] J.-R. Jiang and C.-W. Chu, ''Classifying and benchmarking quantum annealing algorithms based on quadratic unconstrained binary optimization for solving NP-hard problems,'' *IEEE Access*, vol. 11, pp. 104165–104178, 2023.

[4] T. Zaborniak and R. de Sousa, ''Benchmarking Hamiltonian noise in the D-wave quantum annealer,'' *IEEE Trans. Quantum Eng.*, vol. 2, pp. 1–6, 2021.

[5] D. Oku, K. Terada, M. Hayashi, M. Yamaoka, S. Tanaka, and N. Togawa, ''A fully-connected Ising model embedding method and its evaluation for CMOS annealing machines,'' *IEICE Trans. Inf. Syst.*, vol. E102.D, no. 9, pp. 1696–1706, 2019.

[6] H. Kagawa, Y. Ito, K. Nakano, R. Yasudo, Y. Kawamata, R. Katsuki, Y. Tabata, T. Yazane, and K. Hamano, ''High-throughput FPGA implementation for quadratic unconstrained binary optimization,'' *Concurrency Comput., Pract. Exper.*, vol. 35, no. 14, Jun. 2023, Art. no. e6565.

[7] K. Nakano, D. Takafuji, Y. Ito, T. Yazane, J. Yano, S. Ozaki, R. Katsuki, and R. Mori, ''Diverse adaptive bulk search: A framework for solving QUBO problems on multiple GPUs,'' in *Proc. IEEE Int. Parallel Distrib. Process. Symp. Workshops (IPDPSW)*, May 2023, pp. 314–325.

[8] A. Verma and M. Lewis, ''Penalty and partitioning techniques to improve performance of QUBO solvers,'' *Discrete Optim.*, vol. 44, May 2022, Art. no. 100594.

[9] 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.

[10] T. Morstyn, ''Annealing-based quantum computing for combinatorial optimal power flow,'' *IEEE Trans. Smart Grid*, vol. 14, no. 2, pp. 1093–1102, Mar. 2023.

[11] 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.

[12] E. Pelofske, G. Hahn, and H. N. Djidjev, ''Solving larger maximum clique problems using parallel quantum annealing,'' 2022, *arXiv:2205.12165*.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

[24] 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.

[25] M. Mattesi, L. Asproni, C. Mattia, S. Tufano, G. Ranieri, D. Caputo, and D. Corbelletto, "Diversifying investments and maximizing Sharpe ratio: A novel QUBO formulation," 2023, *arXiv:2302.12291*.

[26] 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.

[27] 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.

[28] 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.

[29] 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.

[30] 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.

[31] K. Wils and B. Chen, "A symbolic approach to discrete structural optimization using quantum annealing," 2023, *arXiv:2307.00153*.

[32] R. Sandt and R. Spatschek, "Efficient low temperature Monte Carlo sampling using quantum annealing," *Sci. Rep.*, vol. 13, no. 1, p. 6754, Apr. 2023.

[33] *Digital Annealer Introduction*. Accessed: Jun. 25, 2024. [Online]. Available: https://www.fujitsu.com/ch/de/imagesgig5/da-introduction.pdf

[34] M. Aramon, G. Rosenberg, E. Valiante, T. Miyazawa, H. Tamura, and H. G. Katzgraber, "Physics-inspired optimization for quadratic unconstrained problems using a digital annealer," *Frontiers Phys.*, vol. 7, p. 48, Apr. 2019.

[35] T. Huang, J. Xu, T. Luo, X. Gu, R. Goh, and W.-F. Wong, "Benchmarking quantum(-inspired) annealing hardware on practical use cases," *IEEE Trans. Comput.*, vol. 72, no. 6, pp. 1692–1705, Jun. 2023.

[36] H. Nakayama, J. Koyama, N. Yoneoka, and T. Miyazawa, *Description: Third Generation Digital Annealer Technology*. Tokyo, Japan: Fujitsu Limited, 2021.

[37] S. Matsubara, M. Takatsu, T. Miyazawa, T. Shibasaki, Y. Watanabe, K. Takemoto, and H. Tamura, "Digital annealer for high-speed solving of combinatorial optimization problems and its applications," in *Proc. 25th Asia South Pacific Design Autom. Conf. (ASP-DAC)*, Jan. 2020, pp. 667–672.

[38] O. Şeker, 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.

[39] K. Katayama, N. Yoneoka, K. Kanda, H. Tamura, H. Nakayama, and Y. Watanabe, "Digital annealing engine for high-speed solving of constrained binary quadratic problems on multiple GPUs," in *Proc. IEEE Int. Conf. Consum. Electron. (ICCE)*, Jan. 2024, pp. 1–6.

[40] *Data for the Subset Sum Problem*. Accessed: Jun. 25, 2024. [Online]. Available: https://people.sc.fsu.edu/~jburkardt/datasets/subset_sum/subset_sum.html

[41] 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.

[42] *Meta-Analytics: Max Cut Benchmarks*. Accessed: Jun. 25, 2022. [Online]. Available: http://meta-analytics.net/index.php/max-cut-benchmarks/

[43] *D-Wave Decomposer*. Accessed: Jun. 25, 2024. [Online]. Available: https://docs.ocean.dwavesys.com/en/stable/docs_hybrid/reference/decomposers.html

[44] *Network Repository: A Scientific Network Data Repository With Interactive Visualization and Mining Tools*. Accessed: Jun. 25, 2024. [Online]. Available: https://networkrepository.com/index.php

[45] 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.

[46] K. Mahfouz, S. Ali, M. A. Al-Betar, and M. A. Awadallah, "Solving 0–1 knapsack problems using sine-cosine algorithm," in *Proc. Palestinian Int. Conf. Inf. Commun. Technol. (PICICT)*, Sep. 2021, pp. 45–51.

[47] *Google OR-Tools*. Accessed: Jun. 25, 2024. [Online]. Available: https://developers.google.com/optimization

[48] *Graph Coloring Instances*. Accessed: Jun. 25, 2024. [Online]. Available: https://mat.tepper.cmu.edu/COLOR/instances.html#XXDSJ

[49] 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.

[50] M. Meringer, "Fast generation of regular graphs and construction of cages," *J. Graph Theory*, vol. 30, no. 2, pp. 137–146, Feb. 1999.

[51] 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.*, 6, no. 1, pp. 55–75, 2014.

[52] M. T. Lezana, "A Python implementation of the snakes and ladders for solving the Hamiltonian cycle problem using a graphical interface," Univ. Basque Country, Leioa, Spain, Project Description, 2021.

[53] G. Reinhelt. *TSPLIB: A Library of Sample Instances for the TSP (and Related Problems) From Various Sources and of Various Type*. Accessed: Jun. 25, 2024. [Online]. Available: http://comopt.ifi.uniheidelberg.de/software/TSPLIB95

[54] *OR-Library*. Accessed: Jun. 25, 2024. [Online]. Available: http://people.brunel.ac.uk/~mastjjb/jeb/info.html

[55] J. E. Dorband, "Extending the D-wave with support for higher precision coefficients," 2018, *arXiv:1807.05244*.

[56] H. Chen and D. A. Lidar, "Why and when pausing is beneficial in quantum annealing," *Phys. Rev. Appl.*, vol. 14, no. 1, Jul. 2020, Art. no. 014100.

[57] Z. G. Izquierdo, S. Grabbe, H. Idris, Z. Wang, J. Marshall, and E. Rieffel, "Advantage of pausing: Parameter setting for quantum annealers," *Phys. Rev. Appl.*, vol. 18, no. 5, Nov. 2022, Art. no. 054056.

[58] M. Zielewski, K. Takahashi, Y. Shimomura, and H. Takizawa, "Efficient pause location prediction using quantum annealing simulations and machine learning," *IEEE Access*, vol. 11, pp. 104285–104294, 2023.

[59] A. Callison, M. Festenstein, J. Chen, L. Nita, V. Kendon, and N. Chancellor, "Energetic perspective on rapid quenches in quantum annealing," *PRX Quantum*, vol. 2, no. 1, Mar. 2021, Art. no. 010338.

[60] B. Irsigler and T. Grass, "The quantum annealing gap and quench dynamics in the exact cover problem," *Quantum*, vol. 6, p. 624, Jan. 2022.

[61] G. Passarelli and P. Lucignano, "Counterdiabatic reverse annealing," *Phys. Rev. A, Gen. Phys.*, vol. 107, no. 2, Feb. 2023, Art. no. 022607.

[62] E. Pelofske, G. Hahn, and H. Djidjev, "Initial state encoding via reverse quantum annealing and H-gain features," *IEEE Trans. Quantum Eng.*, vol. 4, pp. 1–21, 2023.

**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 in 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.

**YU-CHEN SHU** (Member, IEEE) received the Ph.D. degree in mathematics from National Taiwan University, Taipei, Taiwan, in 2007. He joined the Department of Mathematics, National Cheng Kung University, as an Assistant Professor, in 2011, and got promoted to Associate Professor, in 2015. Currently, he is the Director of the Intelligence-Computing Credit Program. His research interests include partial differential equations and applied mathematics, especially on elliptic problems, fast algorithms, and mathematical modeling.

**QIAO-YI LIN** is currently pursuing the B.S. degree in computer science and information engineering with National Central University, Taoyuan, Taiwan. His research interests include universal quantum algorithms and quantum annealing algorithms.