Solving NP-hard Problems with Quantum Annealing

Jehn-Ruey Jiang
Dept. of Computer Science and Information Engineering, Naitonal Central University
Taoyuan City, Taiwan
jrjiang@csie.ncu.edu.tw

Chun-Wei Chu
Dept. of Computer Science and Information Engineering, Naitonal Central University
Taoyuan City, Taiwan
109522138@cc.ncu.edu.tw

***Abstract*—Quadratic unconstrained binary optimization (QUBO) formulas of quantum annealing (QA) algorithms are classified into four categories. QA algorithms using different QUBO formulas solve specific NP-hard problems as examples of the classification. The NP-hard problems solved are the subset sum, the vertex cover, the graph coloring, and the traveling salesperson problems. The QA algorithms are compared with their classical counterparts in terms of the quality of the solution and the time to the solution. Based on the comparison results, observations and suggestions are given for each QUBO formula category, so that proper actions can be adopted to improve the performance of QA algorithms. Compared with classical algorithms, QA algorithms are competitive in the current noisy intermediate-scale quantum (NISQ) era and beyond.**

Keywords—noisy intermediate-scale quantum, NP-hard problem, smart manufacturing, quantum annealing, quantum computer, quadratic unconstrained binary optimization

I. INTRODUCTION

Smart manufacturing is a hot research topic recently. Many complex problems, such as NP-hard problems, need to be solved in smart manufacturing applications. Studies advocate using quantum computers, instead of traditional classical computers, to solve NP-hard problems. It is expected that emerging quantum computers can soon solve large-scale and complex problems that traditional classical computers cannot solve within a feasible time, which in turn achieves quantum supremacy [1].

This paper focuses on solving NP-hard problems with quantum annealing (QA) algorithms running on a special quantum computer, the D-Wave quantum annealer [2] consisting of thousands of quantum bits (qubits). Each qubit is made of a tiny superconducting metal ring. At extra-low temperatures, these metal rings become superconductors in which the electric current flows in both clockwise and counterclockwise directions. The superconducting metal ring can be considered to be in a quantum superposition state of 0 and 1 at the same time. In addition, two metal rings can be connected by a coupler. The metal rings and couplers, as well as the control circuit managing the magnetic field biases (bias), make up the programmable quantum processor. The latest D-Wave processor of the D-Wave Advantage system has more than 5,000 qubits and 35,000 couplers [2].

The quantum annealing algorithm formulates a problem as an objective function of a quadratic unconstrained binary optimization (QUBO) formula of binary variables [3]. The minimum value of the objective function corresponds to the optimal solution to the problem. Through the QUBO formula, the problem is embedded into a quantum processor for quantum annealing. The sampling procedure is employed to read the low-energy state that corresponds to the optimal solution to the problem.

We classified QUBO formulas of QA algorithms into four categories. QA algorithms utilizing different QUBO formulas for solving specific NP-hard problems are used as examples of classification. The QA algorithms are compared with classical algorithms running on classical computers in terms of performance metrics like the quality of the solution and the time to the solution. From the comparison results, observations and suggestions are given so that proper actions can be adopted to improve the performance of QA algorithms. The comparison results show that solving NP-hard problems with quantum annealing algorithms using QUBO formulas is competitive in terms of the time to the solution and the quality of the solution.

The rest of this paper is organized as follows. Section II introduces some preliminaries. Four QA algorithms using QUBO formulas are elaborated in Section III. The QUBO formula classification is described in Section IV. QA algorithms using QUBO formulas for solving specific problems are compared with their classical counterparts in terms of different performance metrics in Section V. Finally, some concluding remarks are drawn in Section VI.

II. PRELIMINARIES

A. NP-hard Problem

We first introduce the concepts of deterministic algorithms [4] and nondeterministic algorithms [5]. The deterministic algorithm is the normal algorithm that runs on current (or classical) general-purpose computers to find solutions to problems. On the contrary, the nondeterministic algorithm cannot run on any current computers. It is only for theoretical discussions. A nondeterministic algorithm to solve a given problem has two phases: choosing and checking. The choosing phase is nondeterministic. It selects an option out of given options for subsequent checks. The checking phase is deterministic, though. It checks if the select option leads to a correct solution to the problem or not. If so, it returns "success". Otherwise, it returns "unsuccess". Note that if there exist proper options leading to correct solutions, then the choosing phase is assumed to always select one of the proper options.

On the one hand, problems that can be solved by the deterministic algorithm with polynomial time complexity are called P problems. All P problems form a set (class) called P. On the other hand, problems that can be solved by the nondeterministic algorithm with polynomial time complexity are called NP problems. All NP problems form a set (class) called NP. It is believed that P is a subset of NP.

Cook proved that every NP problem can be polynomially reducible to the satisfiability (SAT) problem [6]. According to

Cook's proof, if the SAT problem belongs to P, then all NP problems also belong to P. A problem is called an NP-hard problem if every NP problem is polynomially reducible to it. Therefore, the SAT problem is an NP-hard problem. Many problems are NP-hard problems. For example, Karp [7] introduced 21 NP-hard problems, such as the maximum cut, Hamiltonian cycle, vertex cover, and 0/1 knapsack problems. None of the NP-hard problems has been solved by the deterministic algorithm with polynomial time complexity. It is likely that there is no such algorithm to solve the NP-hard problem. Therefore, NP-hard problems are regarded as very hard or intractable problems.

B. Quantum Annealing

Quantum superposition and entanglement are the two most well-known properties in quantum mechanics. The two properties have been leveraged to compute for solving problems. With the two properties and the quantum tunneling property, quantum annealing is performed for solving optimization problems. Quantum tunneling [8] is a quantum mechanics phenomenon. When a quantum particle encounters an energy barrier, it may pass through the barrier even if its momentum is less than the barrier potential. Quantum annealing [9] is a meta-heuristic that requires the use of a dedicated quantum annealer like the D-Wave quantum computer to solve problems that optimize objective functions. It goes through the whole solution space to find the global optimal solution. Due to the quantum tunneling property, the found solution is not stuck in local optimization, but reaches the global optimization, as shown in Fig. 1.

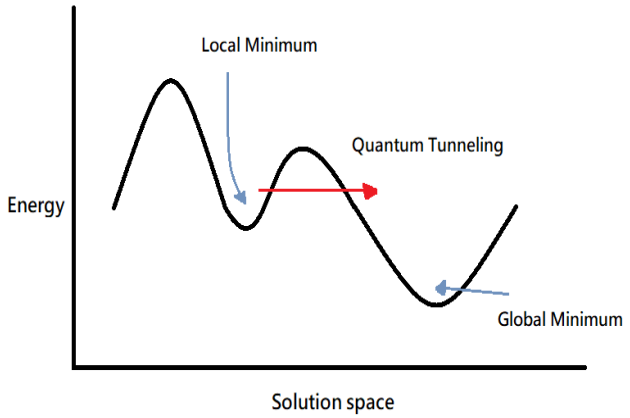

Fig. 1. Illustration of quantum annealing with quantum tunneling.

The quantum annealing algorithm formulates a problem as an objective function of a quadratic unconstrained binary optimization (QUBO) formula f of binary variables, as described below.

$$f(x) = \sum_i \quad Q_{i,i}x_i \; + \sum_{i<j} \quad Q_{i,j}x_i \; x_j \;, \qquad (1)$$

where Q is a symmetric matrix with real-number coefficients, and x is a vector of binary variables. The minimum value of the objective function corresponds to the optimal solution to the problem.

Figure 2 shows the five major steps to designing a quantum annealing algorithm for solving an optimization problem and realizing the algorithm on a quantum annealer like a D-Wave quantum computer. The five steps are as follows.

Step 1. Problem formulation

The optimization problem is formulated as the QUBO formula as a part of a quantum annealing algorithm. Note that some research also formulates the problem as the Ising model [10]. For example, Ref. [11] formulates 21 NP-hard problems as the Ising model. Reference [3] shows that the QUBO formula and the Ising model are equivalent, and they can be transformed into each other easily. Since we focus on the QUBO formula, the optimization problem is assumed to be formulated as a QUBO formula. The formula is in turn transformed into a graph in which a node stands for a binary variable, and each edge between two nodes represents the interaction between the two variables corresponding to the two nodes.

Step 2. Minor embedding

The graph corresponding to the QUBO formula is embedded in the quantum processor or quantum processing unit (QPU) of the quantum annealer, with qubits representing nodes in the graph. Ideally, a qubit must be connected to (coupled with) other qubits. However, due to hardware constraints, a qubit is only connected to a certain number of qubits. To maintain the connectivity of nodes in the graph, multiple qubits are used to represent a node, and the strong chain strength between these bits is set so that these bits can maintain the same value.

When the required number of qubits exceeds the upper limit of the device, the graph corresponding to the original problem needs to be decomposed into subgraphs to be embedded into QPU properly. Currently known graph decomposition (or problem decomposition) methods include the iterative centrality halo method [12], which prioritizes nodes that have significant impacts on the global solution, and the DBK (Decomposition, Bounds, K-core) method, which recursively decomposes a graph into specific-sized subgraphs [13]. Certainly, the performance of quantum annealing is affected by graph decomposition methods [12,13].

Step 3. Compilation

This step sets the final Hamiltonian and the initial Hamiltonian, together forming the Hamiltonian of the entire system. In physics, the Hamiltonian is used to represent the total energy of a system. For a particular state of the system, the Hamiltonian returns the total energy of the system in that state. The system Hamiltonian of a quantum annealer can be expressed as follows.

$$H(t) = \underbrace{A(t)\sum_i \sigma_x^{(i)}}_{\text{Initial Hamiltonian}} + \underbrace{B(t)\left(\sum_i h_i \sigma_z^{(i)} + \sum_{i>j} J_{i,j}\sigma_z^{(i)}\sigma_z^{(j)}\right)}_{\text{Final Hamiltonian}} \qquad (2)$$

In the equation, $\sigma_x^{(i)}$ represents the force of qubit i in the x direction, $\sigma_z^{(i)}$ represents the force of qubit i in the z direction, h_i is the bias of qubit i, and $J_{i,j}$ is the strength of the coupler between qubit i and qubit j. The bias can be used to control the magnitude of the external magnetic field of a qubit, so that the qubit tends to be in state 1 or 0. The coupler strength is used to control the magnitude of the interaction force between two qubits so that the two qubits tend to be of the same state or different states. $A(t)$ and $B(t)$ are energy scaling functions that evolve with the annealing time t.

Note that the final Hamiltonian is set according to the QUBO formula so that the minimum final Hamiltonian corresponds to the optimal solution to the problem. On the other hand, the initial Hamiltonian is set to make every qubit stay in a superposition state for the minimum initial Hamiltonian.

Step 4. Annealing

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

The lower right part of Fig. 2 [9] shows the changing of the energy scaling functions A(t) and B(t) in Equation (2). During the annealing process, the energy scaling function A(t) goes smaller, whereas B(t) goes larger with time t gradually. Consequently, the influence of the initial Hamiltonian becomes more significant, whereas the influence of the final Hamiltonian becomes less significant. At the end of annealing, the system Hamiltonian is mostly affected by the final Hamiltonian.

Step 5. Reading

After the annealing process, the value of the qubit (0 or 1) is read and stored. According to the corresponding relationship between the qubit and the graph node, the solution to the original problem can be derived. If the values of qubits representing the same node are inconsistent, it can be solved by post-processing mechanisms, such as the majority vote, to determine what the value of the node is.

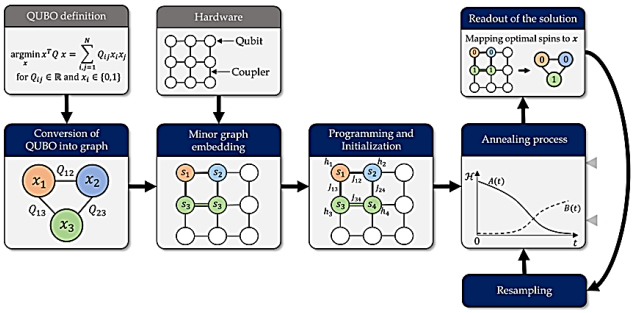

Fig. 2. Workflow of the QA algorithm (adapted from Ref. [9]).

III. QA Algorithms Solving NP-Hard Problems

This section presents four QA algorithms solving four NP-hard problems including the subset sum (SS) problem, the vertex cover (VC) problem, the graph coloring (GC) problem, and the traveling salesperson problem (TSP). The QA algorithms are elaborated on one by one below.

A. QA algorithm solving the SS problem

The SS problem is defined as follows.

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

The QA algorithm solving the SS problem has the following QUBO formula.

$$H(x) = \left(\sum_{i=1}^{n} s_i \; x_i \; - T \right)^2 \tag{3}$$

In the above QUBO formula, s_i is an integer in S, $x_i = 1$ stands for that s_i is in S, and $x_i = 0$ stands for that s_i is not in S, where $1 \le i \le n$.

B. QA algorithm solving the VC problem

The VC problem is defined as follows. Given an undirected graph $G=(V, E)$ with the vertex set V and the edge set E, the vertex cover (VC) problem 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 QA algorithm solving the VC problem has the following QUBO formula.

$$H(x) = A \sum_{(u,v)\in E} (1 - x_u)(1 - x_v) + B \sum_{v \in V} x_v \tag{4}$$

In the above QUBO formula, $x_u = 1$ (resp., $x_u = 0$) stands for that u is (resp., is not) in V, and $x_v = 1$ (resp., $x_v = 0$) stands for 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.

C. QA algorithm solving the GC problem

The GC problem is defined as follows. Given a chromatic number n, and an undirected graph $G=(V, E)$ with the vertex set V and the edge set E, the graph coloring (GC) problem is to decide if it is possible to color all vertices in V such that for every edge (u, v) in E, u and v have different colors.

The QA algorithm solving the GC problem has the following QUBO formula.

$$H(x) = A \sum_{v \in V} \left(1 - \sum_{i=1}^{n} x_{v,i} \right)^2 + A \sum_{(u,v)\in E} \sum_{i=1}^{n} x_{u,i} x_{v,i} \tag{5}$$

In the above QUBO formula, $x_{v,i} = 1$ stands for that vertex v is colored with color i. The first term and the second term are both constraint terms whose weights are both A. The first constraint term means that every vertex should only be colored with only one color. The second constraint term means that adjacent vertices should be colored with different colors.

D. QA algorithm solving TSP

The TSP is defined as follows. Given a directed graph $G=(V, E)$ with the vertex set V of n vertices and the edge set E, the TSP is to find a tour that first visits an arbitrary vertex v, then sequentially visits every vertex exactly once, and at last visits vertex v, such that the sum of weights of edges included in the tour is minimized.

The QA algorithm solving the TSP has the following QUBO formula.

$$H(x) = A\sum_{v=1}^{n}\left(1-\sum_{j=1}^{n} x_{v,j}\right)^2 + A\sum_{j=1}^{n}\left(1-\sum_{v=1}^{n} x_{v,j}\right)^2 + A\sum_{(u,v)\notin E}\sum_{j=1}^{n} x_{u,j}x_{v,j+1} + B\sum_{(u,v)\in E} W_{u,v}\sum_{j=1}^{n} x_{u,j}x_{v,j+1} \quad (6)$$

In the above QUBO formula, $x_{v,j} = 1$ stands for that vertex *v* is the *j*th vertex to be visited. The first three terms are all constraint terms whose weights are *A*. The first constraint term indicates that each vertex must only be visited once, the second term means that only one vertex must 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 must exist an edge (*u*, *v*). The fourth term is an optimization term, in which $W_{u,v}$ is the weight associated with the edge (*u*, *v*).

IV. QUBO Formula Classification

By investigating many QA algorithms, including the above-mentioned QA algorithms solving the SS problem, the VC problem, the GC problem, and the TSP, the QUBO formula of QA algorithms can be classified into four categories according to two criteria.

The two classification criteria are shown below.

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

Several QUBO formulas meet CC1 and maintain a linear relationship between the number of QUBO variables and one input variable of problems, whereas other QUBO formulas do not. For example, the QUBO formulas associated with the SS and the VC problems meet CC1. The QUBO formula associated with the GC problem does not meet CC1. Its number of variables is $n \times |V|$, where *n* is the given chromatic number and $|V|$ is the cardinality of the vertex set *V*. The QUBO formula associated with the TSP does not meet CC1, either; its number of variables is $|V|^2$, where $|V|$ is the cardinality of the vertex set *V*.

(2) Classification criterion 2 (CC2): Does the QUBO formula have both the constraint term and the optimization term?

Several QUBO formulas meet CC2 and have both the constraint term and the optimization term. However, other QUBO formulas do not meet CC2 and have either the constraint term or the optimization term. For example, the QUBO formulas associated with the VC problem and the TSP have both the constraint term and the optimization term. However, the QUBO formulas associated with the SS problem and the GC problem have only the constraint term.

According to the two criteria CC1 and CC2, the QUBO formulas can be classified into four categories. Table I shows the four QUBO formula categories and problem examples each of which can be solved by the QA algorithm using a category of QUBO formulas.

TABLE I. FOUR CATEGORIES OF QUBO FORMULAS

CC2 \ CC1	Yes	No
No	Category 1 (e.g., the QUBO formula for the SS problem)	Category 3 (e.g., the QUBO formula for the GC problem)
Yes	Category 2 (e.g., the QUBO formula for the VC problem)	Category 4 (e.g., the QUBO formula for the TSP)

V. QA Algorithm Performance Comparisons

This section presents the performance comparisons of four QA algorithms and classical algorithms for solving the SS problem, the VC problem, the GC problem, and the TSP. Note that the four QA algorithms use exactly four different categories of QUBO formulas.

The performance comparison experiments are conducted by connecting to the D-Wave Advantage quantum annealer through the Amazon Braket service and the D-Wave Leap service for running the four QA algorithms mentioned above. The performance information of the fastest (lowest time-complexity) classical algorithms solving the same problems is derived from research papers in the literature. Certainly, the QA algorithms and the classical algorithms employ the same problem instances for the sake of fair comparisons.

A. Comparisons of algorithms solving the SS problem

The public problem instances, p01, P01,…, p07, derived from Ref. [14] are used for performance comparisons. Every integer element in set *S* of the SS problem is between 5 and 30, and the target integer *T* is between 20 and 200. Note that there exists a solution to every problem instance. The classical comparative counterpart is a dynamic programming algorithm [14].

As shown in Table II, both QA and classical algorithms find the solutions (indicated by Y) to all problem instances except for the p03 instance, in which set *S* contains many large integers. That is to say, they have the same quality as solutions. QA algorithms may have shorter execution times, either in terms of the total annealing time (AT) or QPU time (QPUT). For problem instances p02 and p03, the QA algorithm is faster than the classical algorithm by a factor of around 30, and 130, respectively. Note that below blue with a superscript =, red with a superscript >, and green with a superscript < are used to indicate that the QA algorithms' experimental results or solutions (Sol) are equal to, better than, and worse than those of classical algorithms, respectively. The time unit is "second".

TABLE II. PERFORMANCE COMPARISONS OF ALGORITHMS SOLVING THE SS PROBLEM

SS Prob.	p01	p02	p03	p04	p05	p06	p07
CA-Sol	Y	Y	N	Y	Y	Y	Y
QAA-Sol	$Y^{=}$	$Y^{=}$	$N^{=}$	$Y^{=}$	$Y^{=}$	$Y^{=}$	$Y^{=}$
CA-Time	0.024	3.463	123.466	0.029	0.051	0.008	0.027
QAA-AT	$0.093^{<}$	$0.117^{>}$	$0.952^{>}$	$0.134^{<}$	$0.097^{<}$	$0.065^{<}$	$0.118^{<}$
QAA-QPUT	$0.077^{<}$	$0.031^{>}$	$0.323^{>}$	$0.066^{<}$	$0.057^{<}$	$0.051^{<}$	$0.081^{>}$

B. Comparisons of algorithms solving the VC problem

The public problem instances, p-hat300-1, keller4, brock400-2, keller5, DSJC500.5, C1000.9, and keller6, derived from DIMACS (Center for Discrete Mathematics and

Theoretical Computer Science) challenge [15] are used for the performance comparisons. The number of vertices in V of the VC problem is between 300 and 1000. The classic comparative counterpart is a branch-and-bound algorithm implemented in Java and complied with JDK 1.7 to run on a platform of CentOS atop a 3.1-GHz CPU with 64 GB memory [16].

As shown in Table III, QA and classical algorithms can find equally good solutions for most problem instances. However, QA algorithms may have worse solutions than classical algorithms for some problem instances. Furthermore, QA algorithms may be faster or slower than the classical algorithms for some problem instances.

TABLE III. PERFORMANCE COMPARISONS OF ALGORITHMS SOLVING THE VC PROBLEM

VC Prob.	p-hat300-1	keller4	brock400-2	keller5	DSJC500.5	C1000.9	keller6
CA-Sol	292	160	371	745	487	932	3298
QAA-Sol	292$^{=}$	160$^{=}$	371$^{=}$	749$^{<}$	487$^{=}$	932$^{=}$	3305$^{<}$
CA-Time	0.560	0.006	0.935	2.380	177.790	0.498	186.540
QAA-AT	53$^{<}$	37.22$^{<}$	34.13$^{<}$	49.2$^{<}$	47.797$^{>}$	20.756$^{<}$	171.7$^{>}$
QAA-QPUT	0.126$^{>}$	0.117	0.121$^{>}$	0.213$^{>}$	0.178$^{>}$	0.107$^{>}$	0.268$^{>}$

C. Comparisons of algorithms solving the GC problem

The public problem instances, R125.1, DSJC125.1, DSJC125.5, R250.1, DSJC250.1, DSJC250.5, DSJC500.1, and le450_15d, derived from DIMACS challenge [17] are used for the performance comparisons. The number of vertices in V of the GC problem is between 125 and 450. The classic comparative counterpart is a memetic algorithm combining the teaching-learning concept and the tabu-search concept [18]. The algorithm is implemented and run on a computer with AMD Operon 6376 CPU and 64 GB memory.

As shown in Table IV, the QA algorithm cannot find solutions for several problem instances, whereas the classical algorithm can find solutions for all problem instances. Furthermore, the QA algorithm may be faster or slower than the classical algorithm for some problem instances.

TABLE IV. PERFORMANCE COMPARISONS OF ALGORITHMS SOLVING THE GC PROBLEM

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

D. Comparisons of algorithms solving the TSP

The public problem instances, br17, ftv33, ftv35, gr17, gr21, p43, ry48p, and kro124p, derived from tsplib [19] are used for the performance comparisons. The number of cities in the TSP is between 17 and 124. The classic comparative counterpart is the algorithm in Google OR-Tools [20].

As shown in Table V, the QA algorithm and the classical algorithm find good solutions for a problem instance, whereas the QA algorithm has worse solutions than the classical algorithm for other problem instances. Furthermore, the QA algorithm is slower than the classical algorithm for all problem instances in terms of total QA time. The former is faster than the latter for two problem instances in terms of the QPU time.

TABLE V. PERFORMANCE COMPARISONS OF ALGORITHMS SOLVING THE TSP

TSP	br17	ftv33	ftv35	gr17	gr21	p43	ry48p	kro124p
CA-Sol	39	1355	1584	2085	2707	5635	14682	41232
QAA-Sol	39$^{=}$	1686$^{<}$	1990$^{<}$	2123$^{<}$	2909$^{<}$	5679$^{<}$	17883$^{<}$	105918$^{<}$
CA-Time	0.027	0.194	0.166	0.055	0.06	0.15	0.261	1.666
QAA-AT	35.3$^{<}$	119.9$^{<}$	128.8$^{<}$	61.0$^{<}$	90.3$^{<}$	70.6$^{<}$	130.7$^{<}$	421.1$^{<}$
QAA-QPUT	0.121$^{<}$	0.173$^{>}$	0.121$^{>}$	0.213$^{<}$	0.198$^{<}$	0.243$^{<}$	0.244$^{>}$	0.476$^{>}$

By the performance comparisons of QA algorithms using different categories of QUBO formulas and classical algorithms for solving different problems, we have the following observations and suggestions. The QA algorithms with the category-1 QUBO formula category are likely to have better performance than classical algorithms. This is because the category-1 QUBO formula usually has fewer QUBO variables and has only the constraint term. The weight assignments of the constraint term(s) and the optimization term(s) affect the performance of QA algorithms using category-2 QUBO formulas. The performance can be improved by carefully assigning proper weights to the two terms. The performance of QA algorithms using category-3 QUBO formulas can be improved by adjusting the quantum annealing time. The category-4 QUBO formulas are the most complex. The performance of QA algorithms using category-4 QUBO formulas can be improved by adopting proper minor embedding procedures and problem partition mechanisms.

VI. CONCLUSION

QA algorithms using QUBO formulas are used to solve NP-hard problems, namely the SS problem, the VC problem, the GC problem, and the TSP. The QUBO formulas used are classified into four categories according to two classification criteria. The two criteria are as follows: (1) “Does the number of QUBO variables have a linear relationship with one input parameter of the problem?” and (2) “Does the QUBO formula have both the constraint term and the optimization term?”

The QA algorithms are compared with their classical counterparts in terms of the quality of the solution and the time to the solution. Compared with classical algorithms, QA algorithms usually have comparatively good solutions to problems and faster or slower than classical algorithms. Based on the comparison results, suggestions are given for improving the performance of QA algorithms using different categories of QUBO formulas. QA algorithms are expected to have superior performance in the future when we go beyond the current noisy intermediate-scale quantum (NISQ) era [21], in which quantum devices have only a moderate number of error-prone qubits.

In the future, we plan to investigate more QA algorithms using different QUBO formulas to solve more problems, such as the maximum cut, the 0/1 knapsack, the Hamiltonian cycle, and the job shop scheduling problems. We also plan to apply the QUBU formulas to developing algorithms for different machines that are similar to the quantum annealer, such as the digital annealer (DA) [22] and the coherent Ising machine (CIM) [23], to see if the DA or the CIM can obtain results with better performance with the help of QUBO formulas.

REFERENCES

[1] F. Arute, K. Arya, R. Babbush, D. Bacon, J. C. Bardin, R. Barends, ... , and J. M. Martinis, "Quantum supremacy using a programmable superconducting processor," Nature, 574(7779), 505-510, 2019.

[2] D-Wave system, url: https://www.dwavesys.com/solutions-and-products/systems/, last accessed in October 2022.

[3] F. Glover, G. Kochenberger, and Y. Du, "A tutorial on formulating and using QUBO models," arXiv preprint arXiv:1811.11538, 2018.

[4] D. E. Knuth, Art of computer programming, volume 2: Seminumerical algorithms, Addison-Wesley Professional, 2014.

[5] R. C. T. Lee, R. C. Chang, Y. T. Tsai, and S. S. Tseng, Introduction to the design and analysis of algorithms, McGraw-Hill, 2005.

[6] S. A. Cook, "The complexity of theorem-proving procedures," In Proceedings of the third annual ACM symposium on Theory of computing. pp. 151-158, 1971.

[7] R. M. Karp, "Reducibility among combinatorial problems," In Complexity of computer computations, pp. 85-103, 1972.

[8] M. Razavy, Quantum theory of tunneling, World Scientific, 2013.

[9] S. Yarkoni, E. Raponi, S. Schmitt, and T. Bäck, "Quantum annealing for industry applications: introduction and review," arXiv preprint arXiv:2112.07491, 2021.

[10] S. S. Wald, Thermalisation and relaxation of quantum systems, Doctoral Dissertation, Université de Lorraine, 2017.

[11] A. Lucas, "Ising formulations of many NP problems," Frontiers in Physics, 5, 2014.

[12] G. Bass, M. Henderson, J. Heath, and J. Dulny, "Optimizing the optimizer: decomposition techniques for quantum annealing," Quantum Machine Intelligence, 3(1), pp. 1-14, 2021.

[13] E. Pelofske, G. Hahn, and H. N. Djidjev, "Solving larger optimization problems using parallel quantum annealing," arXiv preprint arXiv:2205.12165, 2022.

[14] Data for the subset sum problem, url: https://people.sc.fsu.edu/~jburkardt/datasets/subset_sum/subset_sum.html, last accessed in Oct. 2022.

[15] Network repository: a scientific network data repository with interactive visualization and mining tools, url: https://networkrepository.com/index.php, last accessed in Oct. 2022.

[16] L. Wang, S. Hu, M. Li, and J. Zhou, "An exact algorithm for minimum vertex cover problem," Mathematics, 7(7), 603, 2019.

[17] Graph coloring instances, url: https://mat.tepper.cmu.edu/COLOR/instances.html#XXDSJ, last accessed in Oct. 2022.

[18] T. Dokeroglu, and E. Sevinc, "Memetic teaching-learning-based optimization algorithms for large graph coloring problems," Engineering Applications of Artificial Intelligence, 102, 104282, 2021.

[19] G. Reinhelt, TSPLIB: a library of sample instances for the TSP (and related problems) from various sources and of various types, url: http://comopt. ifi. uniheidelberg. de/software/TSPLIB95, last accessed in Oct. 2022.

[20] Google OR-Tools, url: https://developers.google.com/optimization, last accessed in Oct. 2022.

[21] J. Preskill, "Quantum computing in the NISQ era and beyond," Quantum, 2, 79, 2018.

[22] O. Şeker, N. Tanoumand, and M. Bodur, "Digital annealer for quadratic unconstrained binary optimization: a comparative performance analysis," Applied Soft Computing, 127, 109367, 2022.

[23] T. Honjo, T. Sonobe, K. Inaba, T. Inagaki, T. Ikuta, Y. Yamada, ... , and H. Takesue, "100,000-spin coherent Ising machine," Science Advances, 7(40), 2021.