Qobra: Fast Verification of Transactional Serializability with Quantum Annealing

Natsuki Hamada
Keio University
Fujisawa, Kanagawa, Japan
hnatsu@sfc.keio.ac.jp

Kazuhiro Saito
KDDI Research, Inc.
Saitama, Japan
ku-saitou@kddi-research.jp

Hideyuki Kawashima
Keio University
Fujisawa, Kanagawa, Japan
river@sfc.keio.ac.jp

***Abstract*—Serializability is a standard to guarantee the correct execution of database operations. Since cloud databases do not always guarantee serializability, users must check for serializability on their own. However, the internal of cloud databases is a black box for users, making it difficult for them to make a judgment. This is a combinatorial optimization problem called the "black-box serializability problem", which is a satisfiability problem known to be NP-complete. Previous research has proposed an architecture that solves this problem using the SMT solver, a general-purpose solver for the satisfiability problem. Still, as the number of transactions increases, the search space will expand exponentially, so it becomes challenging to determine serializability. On the other hand, quantum annealing is excellent for fast-solving combinatorial optimization problems and can be applied to this satisfiability problem. This paper proposes a fast solver for the black-box serializability problem using quantum annealing. The evaluation results show that the proposed method is 751-890 times faster than state of the art method.**

***Index Terms*—black-box serializability problem, quantum annealing, combinatorial optimization problem**

I. Introduction

A. Motivation

Serializability [23] is a gold standard in database management and transaction processing that ensures that the execution of multiple transactions maintains the consistency and integrity of the database. It guarantees that the end result of concurrent transactions is equivalent to a serial execution of those transactions, even though they might be executed concurrently. In simpler terms, serializability ensures that transactions are executed in a way that preserves the order of operations as if they were executed one after the other, even though they are actually executed concurrently. This helps prevent data anomalies, inconsistencies, and conflicts arising when multiple transactions access and modify the same data simultaneously.

In recent years, the demand for cloud databases has been increasing, and many cloud databases have been developed. However, some databases, such as Amazon DynamoDB [2], Aurora [1], and Azure CosmosDB [3], do not guarantee serializability. Serializability violations can also occur due to internal corruption, which may arise from misconfiguration, misoperation, compromise, or adversarial control at any level of the execution stack. This means that operations executed by the user may be executed in a different order in the database from the original intention of the user.

This paper is based on results obtained from "Research and Development Project of the Enhanced Infrastructures for Post-5G Information and Communication Systems" (JPNP20017), commissioned by the New Energy and Industrial Technology Development Organization (NEDO) and JSPS KAKENHI Grant Number 22H03596.

B. Problem

However, it is difficult for users (developers or administrators) to check whether their operations on the database and the results of those operations in the cloud database are correct. Even if the user knows the protocols for transaction processing inside the database, it is difficult to check whether his/hers operations are ordered based on serializability. This is because the problem is a satisfiability problem known as the "black-box serializability problem", which is NP-complete [10]. The greater the number of transactions to be executed, the greater the number of possible combinations, and the more difficult it is to execute on a classical computer.

Cobra, an existing study, is an architecture proposed to solve this problem [21]. Cobra can determine the serializability of operations given by the user without requiring any protocol mechanism inside the database. The system for solving the black-box serializability problem in Cobra consists of three steps. (1) Create a graph whose vertices are transactions from user operations, (2) Reduce the size of the graph using the proposed three pruning methods, and (3) Use the Satisfiability Modulo Theory (SMT) solver used GPU to determine whether a given operation is serializable or not.

The larger the number of transactions and dependencies in the workload, the more time is required for part (3), where the SMT solver is used to solve the problem. Although the SMT solver is considered an excellent method for solving the satisfiability problem, it is expected that the problem-solving part can be speeded up by applying a more suitable method for this problem.

C. Approach

Quantum annealing (**QA**) is expected to efficiently solve combinatorial optimization problems that are difficult to compute in polynomial time using a classical computer. QA is a quantum algorithm that uses quantum tunneling to obtain a global minimum solution theoretically [20]. QA can be applied to a satisfiability problem depending on the formulation of the constraints.

While there has been prior work that applies QA to the real problems (traffic flow modeling [17], air traffic management [19], multi-car paint shop [24], and nurse scheduling [11], [15]), ours is the first work that clarifies the effectiveness of the QA with a real (non-pseudo) quantum machine for black-box serializability problem with comparison to classical methods to the best of our knowledge.

We applied QA instead of SMT to the solver part of Cobra to solve the black-box serializability and succeeded in solving the problem faster. We call this "Qobra". Cobra has two methods for determining serializability: one-shot verification and round verification. One-shot verification is a method in which the entire history is given to Cobra at once. In round verification, the history is divided and given to Cobra, and it is solved while reducing the size of the graphs by using the proposed garbage collection. Note that Qobra is implemented only for one-shot verification.

The evaluation was based on three analysis criteria. The three criteria are performance, reliability, and scaling. The performance of the QA execution results was evaluated in terms of the probability of obtaining a feasible solution and its execution time. For reliability, QA is an approximate solution method, and it is not always possible to obtain an exact solution because the solution changes every execution. Therefore, to make a fair comparison with the existing SMT method, an exact solution method, we evaluated QA using TTS, which will be discussed in section V-C. Finally, scaling evaluations were conducted to evaluate how the execution times of QA and existing methods change as the number of transactions increases, and discussed the superiority of QA.

Our contributions are that (1) we formulated the solver part of Cobra for the black-box serializability problem by QUBO, and (2) by changing the solver part of Cobra from SMT solver to QA, we succeeded in solving the problem by obtaining a feasible solution 99% and the time is 751 to 890 times faster than the existing methods.

D. Organization

The rest of this paper is organized as follows: Section II describes the preliminary background knowledge about quantum annealing and serializability. Section III introduces state of the art method, Cobra. Section IV presents proposed method, applying QA for the black-box serializability problem. Section V describes the experimental environment and evaluates the performance of QA by comparing it with existing methods and QA simulators. Section VI describes related work, and Section VII concludes this paper.

II. Preliminaries

A. Quantum Annealing

Quantum annealing (**QA**) is expected to efficiently solve combinatorial optimization problems that are difficult to compute in polynomial time using a classical computer. QA is a quantum algorithm that uses quantum tunneling to obtain a global minimum solution theoretically [20].

To apply QA for combinatorial optimization problems, we must formulate the problems as the Ising model. The Ising model represents a two-dimensional lattice model of ferromagnetic spins in statistical mechanics.

$$E(\boldsymbol{s}) = \sum_{i,j} J_{ij} s_i s_j + \sum_i h_i s_i \tag{1}$$

Equation (1) represents the Ising model. The N variables (spins) $\boldsymbol{s} = [s_1, \ldots, s_N]$ are binary with $s_i = \{+1, -1\}$. J_{ij} and h_i are the coefficients representing the spin interaction and the value of the local field, respectively. QA can solve the combinatorial optimization problem by formulating it into the Ising model as the minimization of the objective function. Minimizing the Ising model is known to be NP-hard [8].

Since the formulation needs to be intuitive when solving real problems, we convert the spins $\boldsymbol{s_i}$ into binary variables $x_i \in \{0, 1\}$ as follows.

$$x_i = \frac{s_i + 1}{2} \tag{2}$$

To use QA, this is applied to the Ising model in (1) to obtain the following quadratic unconstrained binary optimization (**QUBO**).

$$E(\boldsymbol{x}) = \sum_{i,j} Q_{ij} x_i x_j \tag{3}$$

Therefore, it is possible to apply the black-box serializability problem to QA by formulating the objective function and constraints of the black-box serializability problem to the QUBO in (3). By using (2), the Ising model can be converted into QUBO and vice versa, so we can choose the formulation that is easier to formulate depending on the problem. In this paper, we formulate a black-box serializability problem in QUBO.

B. Black-box Serializabililty Problem

1) Setting: As summarized below, this paper considers within a standard framework under a particular isolation level [23]. First, a unique version of each database key is created when writing to the key, and each transaction reads and writes the key at most once. The history is the set of operations formed by the transaction and has a version. The version order is obtained from within the database and is not exposed to the outside world. A history is said to be serializable if there exists a total order of committed transactions such that executing transactions in a particular order yields the same result as the history.

By looking at the history, it is possible to identify transaction dependencies. A dependency is a relationship in which two transactions execute operations on the same key, and changing the order of execution may change the result. In other words, if this dependency is broken, the criteria for serializability are not satisfied. There are three types of dependencies. The first is a read dependency. This dependency occurs when transaction T_j reads the value written by transaction T_i ($T_i \rightarrow T_j$). The second is a write dependency. Since T_i writes

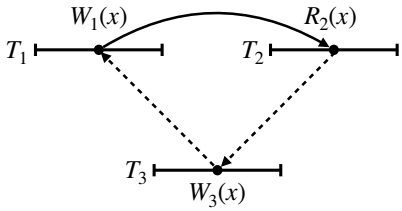


Fig. 1. An easy example of polygraph

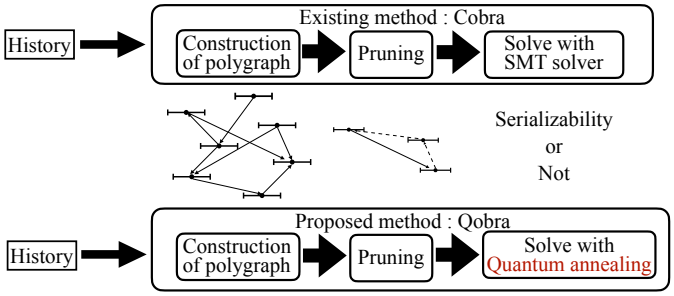


Fig. 2. The architecture of Cobra and Qobra

a key and T_j overwrites it, the order must be fixed. The third is anti-dependency. Since T_i reads the value overwritten by T_j, the order must not be changed.

A serializability graph checks whether serializability is satisfied in a history. The vertices of this graph are the transactions in the history, and the edges are the three dependencies mentioned earlier. Note that the graph does not include executing transactions or transactions that have been aborted. If the graph does not have cycles, then the history is serializable. Conversely, if there is a cycle, the history is not serializable.

The difficulty with this problem is that it does not reveal the order of operations executed by the database, so it is not possible to reveal all of the dependencies. Since version order is not available to us, we need to consider all possible version orders to check whether any dependencies, including a potential set of ones, could be a serializable graph without cycles.

This paper deals with the view serializability among the serializability criteria.

2) Polygraph: A data structure that further extends the serializable graph to store a set of potentially existing dependencies is called a polygraph, and in Cobra, this polygraph plays a significant role.

In a polygraph, a vertex (V) is a transaction, and an edge (E) is a read-only dependency. We add a new set of constraints (C). C stores potential dependencies. The following diagram provides an intuitive explanation.

On the graph in Fig. 1, the vertices are $V = \{T_1, T_2, T_3\}$ and the edges are $E = \{(T_1, T_2)\}$, derived from the reading dependence $W_1(x) \rightarrow R_2(x)$ (solid arrows in the figure). The constraints are represented by pairs of potential edges of $C = \langle (T_2, T_3), (T_3, T_1) \rangle$ (dotted arrows in the figure). Now, the dependency $T_1 \rightarrow T_2$ has been established, and a new transaction T_3 has appeared. The order of $T_1 \rightarrow T_2$ must be kept, so T_3 may be executed before or after them. If it is executed before the dependency, $T_3 \rightarrow T_1 \rightarrow T_2$, and if it is executed after that, $T_1 \rightarrow T_2 \rightarrow T_3$, the constraint C is used to represent these two possible edges.

In the following, polygraphs are defined using mathematical formulas. A polygraph $P = (V, E, C)$ is a set of directed graphs (V, E) called known graphs plus a pair of edges called constraints C.

- V represents all committed transactions in the history.
- $E = \{(T_i, T_j) | T_j \text{ reads from } T_i\}$.
- $C = \{\langle (T_j, T_k), (T_k, T_i) \rangle | T_j \text{ reads from } T_i \wedge (T_k \text{ write to } x) \wedge T_k \neq T_i \wedge T_k \neq T_j\}$

The edge E represents all read dependencies. The C represents potential dependencies that may exist. All transactions that write to the same key for each read dependency in the history occur after the commited read or before the write.

Polygraphs can be used to solve the black-box serializability problem. Constraint C is a pair of edges that can be dependencies, so if no cycle occurs when one of the dependencies (edges) is selected and applied to the known graph, then the graph is serializable. The graph is called a compatible graph if one edge is actually selected from among the constraint pairs of the polygraph and applied to the known graph. The formulation is as follows.

Consider a graph (V', E'). If there exists a polygraph (V, E, C) and $V = V' \wedge E \subseteq E' \wedge \{\forall \langle e_1, e_2 \rangle \in C, (e_1 \in E' \wedge e_2 \notin E') \vee (e_1 \notin E' \wedge e_2 \in E')\}$, then the graph (V', E') is a compatible graph. A given history is serializable if no cycle exists in the compatible graph.

3) Computational Complexity: To check whether a given history is serializable, it is necessary to construct a polygraph $P = (V, E, C)$ from that history and to check whether there are any cycles in the compatible graph (V', E') that can be constructed from the polygraph. The number of compatible graphs that can be created from a polygraph is $2^{|C|}$, since the number of candidates for each pair of constraints C equals the number of candidates for choosing one edge from each pair of constraints C.

III. State of the Art: Cobra

A. Design

Cobra [21] is an architecture proposed to solve the black-box serializability problem. Cobra can judge whether operations given by the user are serializable without requiring any protocol mechanism inside the database. Fig. 2 shows the architecture of Cobra. The system for solving black-box serializability problems in Cobra primarily consists of three parts in Fig.2. (1) First, the operations from the user are combined into a history, and a polygraph is created from that history. (2) Next, the number of dependencies (edges) in the polygraph is reduced using the three pruning methods, while making constraints and checking for cycles of the polygraph. (3) Finally, based on the reduced size of the polygraph, it is

solved using MonoSAT [9], a GPU-based SMT (Satisfiability Modulo Theory) solver, and judged whether it is feasible or not.

Strictly speaking, Cobra has two types of judgments. The first is a one-shot verification in which the polygraph is created without dividing the history. The second is a round verification in which the history is divided, the divided history is verified step by step, and unnecessary transactions are removed from the polygraph using proposed garbage collection. This paper deals only with one-shot verification.

The pruning method in Cobra's second part not only reduces the size of the polygraph but also has a significant role in determining Serializability. Therefore, we briefly introduce below. The three pruning methods are combining write, coalescing constraint, and pruning constraints. When creating a polygraph, vertex (transaction) and edge (dependency) are created first, and constraint is created using the pruning method. To be precise, the second pruning method, Coalescing constraints, starts to create the constraints.

Combining write is a method to reduce dependency by combining read and write operations in database operations into a single array called a chain. It is easy to imagine that a series of operations that reads the value of a data item and then writes it based on that value are very common operations in actual applications. Therefore, this pruning method is very effective for reducing the size of a polygraph.

The second is a coalescing constraint, which means that when edges are connected if there is a constraint whose starting and ending points are equal, it is not added to the constraint. The role of an edge is to detect if a cycle does not exist in the polygraph. Therefore, it is essential that the start and end points of consecutive edges are not equal. Conversely, it is sufficient if this can be confirmed. Therefore, such a constraint does not need to be added.

Finally, the Pruning constraint plays a role not only as a pruning method but also as a cycle detector for polygraphs. To judge serializability, it was necessary to meet two conditions: one of the two edges of the constraint must be selected so that it does not overlap with the other edge, and the selected edge must not be able to cycle. Cycle detection was determined using this pruning method. Each constraint is checked, and if one of the two potential edges is selected and can be cycled, it is removed, and the remaining potential edge adds to edge. If a cycle is generated when both potential edges of the constraint are selected, then we can determine that the history given to Cobra is not serializable. By detecting cycles in the polygraph at this time, we only need to satisfy one condition in the next solver part, which is to select one of the two potential edges of the constraint so that it does not overlap with any other potential edge of the constraint.

I repeat, to determine whether a given history is serializable, it is necessary to find a combination of dependencies represented by the constraints of the polygraph that does not generate cycles. This means that there are two considerations to be made. The first is to check whether a cycle occurs in the polygraph. The second is that only one edge must be selected from a pair of each constraint. In Cobra, the first cycle detection can be done with the pruning method. Cobra uses a solver to determine the second.

B. Solver Part of Cobra

To apply polygraphs to the SMT solver in the solver part, we need to formulate the problem into a satisfiability (SAT) formula. Cobra represents the graph G and the constraints C as follows. Cobra's solver part constructs a boolean variable E_{ij} for each vertex-vertex pair in the graph G. True (or false) means that the explored compatible graph has (or does not have) the edge $(T_i \rightarrow T_j)$. For example, E_{34} being true means that it has the edge $(T_3 \rightarrow T_4)$. For constraint C, each constraint $\langle A, B\rangle$ is a pair of sets of edges, and if we select an edge contained in set A, the edge must not be in set B. On the contrary, if an edge in the set B is selected, the edge must not be in the set A. This can be written as follows.

$$\{(\forall e_a \in A, e_a) \wedge (\forall e_b \in B, \neg e_b)\} \vee \{(\forall e_a \in A, \neg e_a) \wedge (\forall e_b \in B, e_b)\} \quad (4)$$

Finally, by applying this (4) to MonoSAT, a calculation is executed inside MonoSAT, which returns true if there is a solution that satisfies this SAT expression, and false if there is no solution. This true/false result corresponds to whether or not the given history is serializable.

As mentioned earlier, the size of E_{ij} equals the number of vertex-vertexes. In other words, the number of variables depends on the number of transactions, since a vertex in a polygraph is a transaction. Note that the number of variables in solving with SMT solver depends on the number of transactions. As the number of variables increases, execution time increases. Reducing the number of variables will decrease the execution time.

IV. Proposal: Qobra

In this section, we explain Qobra and why and how we applied quantum annealing to the solver part of Cobra.

The larger the number of transactions and dependencies in the workload, the more time is required to solve part with the SMT solver. Although the SMT solver is considered an excellent method for solving the satisfiability problem, it is expected that the solver part can be speeded up by applying a more suitable method for this problem.

Quantum annealing (**QA**) is expected to efficiently solve combinatorial optimization problems that are difficult to compute in polynomial time using a classical computer. It has been applied to real-world combinatorial optimization problems, and excellent results have been reported [11], [15], [17], [19], [24]. The Black-box serializability problem is a satisfiability problem, but satisfiability problem can be applyed QA depending on formulation of QUBO or Ising model. This is why QA is expected to improve the system's performance.

A. Problem Setting

To apply QA to the black-box serializability problem, we implemented Cobra's solver part by changing the solver to QA, as shown in Fig. 2. We name this system "Qobra". Fig. 2 is a simple illustration for Qobra. In Qobra, Cobra is used up to the pruning part, and only the solve part is replaced by QA. To solve the Black-box serializability problem by using polygraphs, the following two conditions had to be satisfied. First, one potential edge must be selected from each pair of potential edges in constraint C defined in section II-B2, and they must be unselected edges of another pair of edges in C. Second, no cycle is created on the selected potential edge and the edge of E. As explained in section III-A, Cobra's pruning part not only performs polygraph pruning but also cycle detection. Therefore, the second condition, cycle detection, is not a condition that must be satisfied by the solver part. Therefore, the only condition that must be satisfied is that one edge is selected from each pair of potential edges in constraint C and that they are unselected edges of another pair of edges in C.

B. QUBO Formulation for Cobra's Solver Part

Equation (4) cannot be applied to QA directly, because it needs to be formulated in the QUBO in (3) or the Ising model to be applied to QA. The SAT formula in (4) contains many logical products, equivalent to multiplying variables by each other. The Ising model and QUBO cannot be applied to QA directly because the variables are allowed only up to the second order. Furthermore, as mentioned in the previous section, the number of transactions in (4) depends on the number of variables, and another way to reduce the number of variables would lead to a speed-up. Therefore, we reduced the number of variables by workload by taking dependency as the variable instead of all edges.

To apply the problem to QA, it is necessary to formulate the problem in the form of QUBO in (3) or Ising model in (1). In this problem, we choose QUBO because the formulation of QUBO is more intuitive than that of the Ising model.

Table I shows the definitions of the symbols used in formulation. They are entirely different from the symbols used in Cobra's SAT formulation.

The graph on the left of Fig. 3 is an example of a polygraph. The vertices are $V = \{T_1, T_2, T_3, T_4\}$ and the constraint C is two pairs, blue and red, such that $C = \{\text{blue}, \text{red}\}$. blue is the pair (e_{23}, e_{31}), red is the pair (e_{43}, e_{31}). Therefore, the set of all edges is $E = \{e_{23}, e_{31}, e_{43}\}$ and the decision variables are $x_{e_{23}}, x_{e_{31}}, x_{e_{43}}$.

What we want to formulate is that when selecting an edge from a pair of constraint edges, only one edge must be selected for all pairs. In the example shown in Fig. 3, there are two solutions $(x_{e_{23}}, x_{e_{31}}, x_{e_{43}}) = (1, 0, 1), (0, 1, 0)$ that satisfy this condition. We need a formulation in which this becomes the solution. The table on the right side of Fig. 3 is G_{ce} generated from the polygraph on the left. This table enables the formulation.

$$\forall c \in C, \quad \sum_{e \in E} (G_{ce} \cdot x_e) = 1 \tag{5}$$

Equation (5) is a general formulation. If this formula holds for all c, then the condition that only one edge must be chosen in every pair of constraint edges can be satisfied. If we calculate $G_{ce} \cdot x_e$, only those edges which are in one constraint c and are selected take the value 1. Therefore, by satisfying the (5), we have only one edge for each constraint c.

$$\mathbf{H} = \sum_{\mathbf{c} \in \mathbf{C}} \left[\left\{ \sum_{\mathbf{e} \in \mathbf{E}} (\mathbf{G_{ce}} \cdot \mathbf{x_e}) \right\} - \mathbf{1} \right]^{\mathbf{2}} \tag{6}$$

Equation (6) is an extension of (5) to QUBO. It plays the role of a constraint term where the $\{\sum_e (G_{ce} \cdot x_e)\} - 1$ part is 0 and the rest is 1 if the conditions of the (5) are satisfied. If the sum of all these constraints is non-zero, it means that some constraint has been violated and the desired condition is reflected. Since QA is an algorithm that calculates the minimum value given a QUBO, the solution is derived so that this constraint term becomes zero. Not limited to quantum annealing, if the obtained solution is substituted into (6) and 0 is obtained, then the solution satisfies the constraint. Note that although quantum annealing is an approximate solution method, if (6) is equal to 0, the solution that satisfies the constraint can certainly be obtained.

V. Evaluation

In this section, we describe the experimental results of Qobra in which QA was applied to the solver part of Cobra.

A. Workload

In this paper, we used two benchmarks to evaluate performance of QA.

C-Twitter [4] enables users to tweet new posts, follow and unfollow other users, and view their timelines (the latest tweets of the followed users). This experiment involves 1,000 users, each of whom tweets a 140-character post and follows or unfollows other users based on a Zipfian distribution ($\alpha = 100$).

BlindW-RW measures performance in extreme scenarios, especially when there are many blind writes (i.e. writes that are not preceded by a read of the same key in the same transaction). The benchmark creates 10k keys and executes read-only and write-only transactions with 8 operations each. In this paper, the benchmark uses BlindW-RW (Read Write), in which read-only and write-only transactions are equally divided.

Each workload is logged by Cobra Bench provided by Cobra [6], and the polygraph is created by Cobra Verifier. Then, the polygraph is pruned while the cycle detection is executed and applied to QA. The details of each workload, such as the number of C constraints, are listed in the table II.

TABLE I
SYMBOLS FOR FORMULATION

Symbol	Description
N	Set of transaction ID
V	Set of transactions T_n where $n \in N$.
C	Set of constraint pairs.
E	Set of edges contained in $c \in C$ without duplication.
G_{ce}	Binary variable that satisfied $\sum_{e \in E} G_{ce} = 2$. 1 if the constraint $c \in C$ contains an edge $e \in E$, 0 otherwise
x_e	Decision variable which take 1 if the edge $e \in E$ is selected, and 0 otherwise.

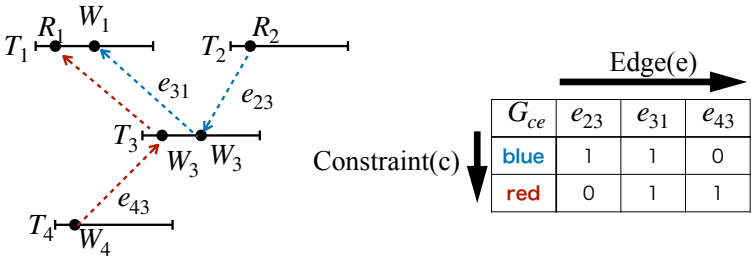


Fig. 3. Example of constructing G_{ce} from a polygraph

TABLE II
DETAILS OF EACH WORKLOAD

Workload	#transactions	#constraints after pruning
BlindW-RW	101	8
BlindW-RW	195	29
C-Twitter	9,991	62

TABLE III
DETAILS OF `num_sweeps`

Workload	#transactions	`num_sweeps`
BlindW-RW	101	5-485 increase by 30
BlindW-RW	195	15-495 increase by 30
C-Twitter	9,991	50-950 increase by 50

B. Methods and Execution Environment

The three methods compared in this work are as follows.

Quantum annealing (D-Wave Advantage). We used the Advantage system 4.1 [5], a hardware implementation of the quantum annealing machine provided by D-Wave systems as a cloud service. We set two parameters, `num_reads` and `annealing_time`, apart from the default QA settings to improve QA performance. We set `num_reads`=1,000, which is the number of runs of the QA. `annealing_time` sets the duration, in microseconds with a resolution of 0.01 μs for D-Wave Advantage, of quantum annealing time, per read. We set `annealing_time` varying values in 10 increments from 10 to 300 for all workloads. We measured the qpu_sampling_time of Ocean software provided by D-Wave Systems as the execution time of the QA. The qpu_sampling_time includes not only the annealing time but also the post-processing time after annealing in the QPU. Since this is performed for `num_reads` (=1,000), the execution time per run, which is used after section, is calculated by dividing the post-processing time by `num_reads`.

Simulated quantum annealing (SQA). In this paper, we use SQASampler provided by OpenJij [7] as a simulated quantum annealing (SQA). The parameters of SQA are fixed at `beta`=10, `gamma`=1.0, `trotter`=10, `num_reads`=1,000. The other parameters are set to default. SQA also has a parameter `num_sweeps`. The parameter `num_sweeps` indicates the number of sweeps when the transverse magnetic field is lowered during annealing. The greater the value is, the slower the annealing, and the longer the annealing time, the more stable a feasible solution can be obtained. Section V shows how execution results change as this `num_sweeps` is increased. We varied the value of `num_sweeps` to suit each workload, as detailed in Table III.

Satisfiability modulo theories (SMT). Satisfiability (SAT) is a problem to determine whether a constraint term has a solution or not. Satisfiability modulo theories (SMT) is an extended version of SAT that can also deal with the true or false of predicate logic, whereas SAT can only deal with the true or false of a single variable. Unlike local search methods, including QA, SMT solver can ensure that a feasible solution can be obtained once it is executed. MonoSAT [9] are applied in Cobra as a SMT solver. Although MonoSAT is a solver for solving SMT but the formulation in Cobra is SAT, SAT can also be solved by SMT solvers because SMT is the extended version of SAT. MonoSAT is a suitable solver for graph-based problems and solves problems faster by using GPU.

The classical execution environment for QA's library, SQA, and SMT is a p3.2xlarge instance of Amazon EC2, with 8 CPU cores and 64 GB of memory, and the GPU used for the SMT solver is an NVIDIA Tesla V100. The versions of the Python libraries required to run D-Wave Advantage are as follows: dwave-ocean-sdk is version 6.4.1, dimod is 0.12.7,

and pyqubo is 1.4.0. The versions of the Python libraries required to run OpenJij's SQA are as follows: Openjij is version 0.7.3. Cobra, Qobra and MonoSAT are implemented in Java, OpenJDK version is 1.8.0_362, and MonoSAT is version 1.4.0. MonoSAT requires CUDA because it uses a GPU, and the version of CUDA is 10.0.130.

C. Performance Metric: Time To Solution

Because QA and SQA are types of local search method, they cannot be guaranteed to obtain a feasible solution in only one run. SMT, the exact solution method, can definitely determine whether a feasible solution can be obtained in a single run. Therefore, it is unequal to compare QA and SQA, which are approximate methods, with SMT based only on a single execution time. So, we use a metric called time to solution (TTS) to evaluate the comparison between approximate and exact solution methods as equally as possible.

TTS is the time it takes to satisfy the desired condition by executing it a sufficient number of times if the desired condition cannot be obtained with only one run. In this paper, time to solution means the time to obtain a feasible solution. We consider the probability of obtaining a feasible solution when the number of executions is increased. For the probability r_1 of obtaining a feasible solution in a single annealing process, the probability r_m of obtaining a feasible solution in m runs is as follows:

$$r_m = 1 - (1 - r_1)^m. \tag{7}$$

The number of times m required to obtain a feasible solution with probability r_m can be calculated from (7). The TTS is obtained by multiplying m by the execution time of one run τ.

$$TTS(\tau, r_1, r_m) = \tau m = \tau \left\lceil \frac{log(1 - r_m)}{log(1 - r_1)} \right\rceil \tag{8}$$

If the probability of obtaining a feasible solution r_1 is small even if the execution time per execution (τ) is short, the number of executions m of (8) becomes large, and as a result, TTS may become large. On the other hand, if the probability of obtaining a feasible solution is large, even if the execution time per execution is long, the number of executions m can be reduced, and thus TTS can be small.

TTS is used for the approximate solution methods, QA and SQA, while the exact solution method, SMT, uses the measured execution time directly. In addition, TTS obtained by QA and SQA is changed by varying the parameters that depend on the execution time (i.e. `annealing_time` and `num_sweeps`). We also evaluate multiple TTSs on each method by changing the parameters.

The details of QA's TTS calculation are as follows. First, one execution time of QA was measured by qpu_sampling_time with QA executed under `num_reads` = 1,000 while varying the parameter `annealing_time`. We obtained the one execution time, τ, by dividing the time by `num_reads`. Second, we also divided the number of times a feasible solution was obtained into `num_reads`, and this value was defined as the probability of obtaining a feasible solution, r_1. SQA's TTS calculation is the same as for QA's one except that execution time of SQA was measured by varying the parameter `num_sweeps`.

In order to increase the reliability of QA and SQA, which are approximate solution methods, we compare the change in TTS of QA and SQA when the probability of obtaining a feasible solution is larger by increasing r_m to 0.99, 0.999, and 0.9999 in section V-D3.

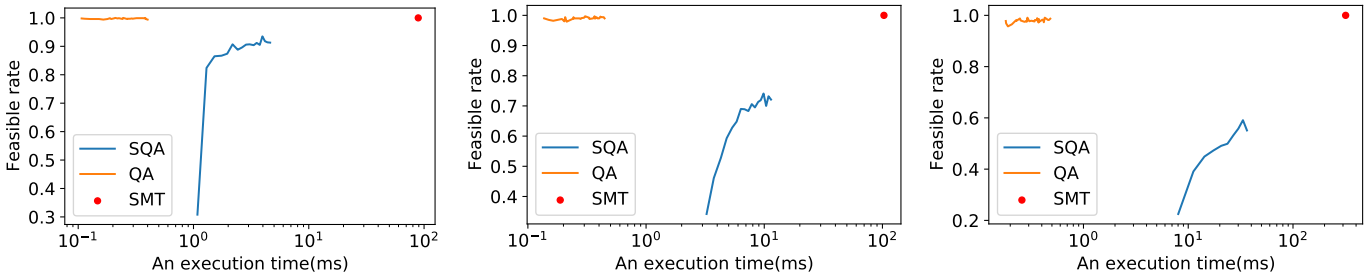
D. Result

The evaluation was based on three analysis criteria. The three criteria are performance, reliability, and scaling.

1) Evaluation of performance: The performance of the QA execution results was evaluated in terms of the probability of obtaining a feasible solution and its execution time. Both are derived directly from the execution results and are indicators of QA performance.

Fig. 4 shows a comparison of the feasible solution rate and execution time by the three methods for the three workloads represented in Table II. The left graph shows a workload, BlindW-RW, with 101 transactions, the middle graph shows a workload, BlindW-RW, with 195 transactions, and the right graph shows a workload, C-Twitter, with 9,991 transactions. The horizontal axis represents an execution time by log scale, and the vertical axis represents the feasible solution rate. Since SMT is an exact solution method, the probability of obtaining a feasible solution is 1. This is represented by the red dot. QA and SQA are represented by orange and blue line graphs, respectively, and represent the feasible solution obtained by varying the parameters (`annealing_time` and `num_sweeps`), which affect the execution time.

For the workload, BlindW-RW, with 101 transactions, QA achieves a feasible solution with a very short execution time of about 0.1 ms with a probability of almost 1. SQA has an execution time of about 1.1-4.6 ms, with a probability of 0.9 from an execution time of 2.8 ms. SMT obtains an exact solution in 89 ms. QA can obtain a feasible solution, almost certainly in a very short execution time. For the workload, BlindW-RW with 195 transitions, QA achieves a solution in 0.1-0.4 ms, and feasible solutions are obtained with a probability of almost 1, even for workloads with an increased number of transitions. SQA has an execution time of 3.2-11.4 ms, and feasible solutions are obtained with a probability of 0.7, starting at around 7.8 ms. SMT achieves an exact solution in 103 ms. On the C-Twitter workload with 9,991 transactions, QA has an execution time of 0.2-0.5 ms, and despite the increased number of transactions, the probability of obtaining a feasible solution is nearly 1, indicating very high performance. SQA achieves execution times of 8-36 ms, with the probability of obtaining a feasible solution increasing to 0.5 from around 20 ms. SMT achieves an exact solution in 318 ms. For all workloads, QA has a shorter execution time than the other two methods, and it achieves a feasible solution with a probability of almost 1.



BlindW-RW
#transactions = 101

BlindW-RW
#transactions = 195

C-Twitter
#transactions = 9,991

Fig. 4. Feasible solution rate with varying execution time

2) Evaluation of Reliability for Each Workload: For evaluation of reliability, QA and SQA are approximate solution methods, and it is not always possible to obtain an exact solution because the solution changes every execution. The evaluation is based on a metric called TTS described in section V-C, which is the number of runs until the desired probability of obtaining a feasible solution is reached. Adopting such a metric not only assures the reliability of QA but also allows for a fairer comparison with the exact solution method. In this section, r_m (target probability) is set to 0.99. In other words, the evaluation is performed under QA and SQA obtain a feasible solution with a probability of 99%.

Fig. 5 shows a graph comparing the three methods in terms of TTS and an execution time for the three workloads represented in Table II. The TTS here is calculated with r_1 in (4) as the probability of obtaining the feasible solution shown in Fig. 3, with τ as an execution time, and with r_m=0.99. The left graph shows the workload of BlindW-RW with 101 transactions, the middle graph shows the workload of BlindW-RW with 195 transactions, and the right graph shows the workload of C-Twitter with 9,991 transactions. The horizontal axis represents execution time by log scale, and the vertical axis represents TTS by log scale. Since SMT is an exact solution method, the probability of obtaining a feasible solution is 1, so TTS is set equal to the execution time. This is represented by the red dots. QA is represented by an orange line graph, and SQA by a blue line graph.

For the workload, BlindW-RW, with 101 transactions, QA has the smallest TTS value of 0.1 ms, SQA has 5.2 ms, and SMT has an execution time of 89 ms. QA is 52 times faster than the TTS of SQA and 890 times faster than the execution time of SMT. For the workload, BlindW-RW, with 195 transactions, the smallest TTS value for QA is 0.14 ms, SQA is 25 ms, and the execution time of SMT is 103 ms. QA yields a TTS that is 185 times faster than the TTS of SQA and 751 times faster than the execution time of SMT. For the workload, C-Twitter, with 9,991 transactions, the lowest TTS obtained with QA is 0.36 ms, and SQA is 112 ms. The execution time of SMT is 318 ms. The TTS of QA is 311 times faster than the TTS of SQA and 883 times faster than the execution time of SMT. Compared to the execution time of SMT with TTS for the metric introduced to fairly compare the approximate solution method with the exact solution method, QA is able to obtain solutions the fastest of all on any workload.

3) Evaluation of Reliability and Scaling: Scaling evaluations were conducted to evaluate how the TTS of QA and existing methods change as the number of transactions increases.

In order to show scaling, Fig. 6 shows the three workloads in Table II sorted by the number of transactions and the value of the smallest TTS in that case. In order to show reliability, Fig. 6 also shows the change in TTS of QA and SQA when r_m (the target value of the probability of obtaining a feasible solution in 8) is increased to 0.99, 0.999, and 0.9999, in the order from left to right. Note that SMT is an exact solution and, therefore, represents a single execution time, so changing r_m does not affect the value of SMT. The horizontal axis is the log scale of the number of transactions. The vertical axis is the log scale of TTS, and all three graphs share the same vertical axis. QA is represented by an orange line graph, SQA by a blue line graph, and SMT by a red point.

First, focus on the figure when r_m=0.99, that is, when a feasible solution is obtained with 99% probability (the left figure in Fig. 6). QA is scaled from 0.10-0.29 ms, and TTS is kept at a very low value even as the number of transactions increases. SQA is scaled from 3.9-112 ms, and as the number of transactions increases, the TTS value becomes very large. For SMT, the execution time ranged from 89-318 ms. As the number of transactions increases, the rate of growth of the execution time also increases, although not as much as for SQA. Next, focus on the figure when a feasible solution is obtained with r_m=0.999 (the middle figure in Fig. 6). The TTS of QA is 0.19-0.36 ms, which is still fast despite the increase in the number of transactions. Compared to the case with r_m=0.99, the TTS is generally slower but still fast enough. The TTS for SQA is 5.2-157 ms, which is a higher rate of increase than the other two methods. Finally, we turn our attention to the case of TTS for which a feasible solution can be obtained with r_m=0.999 (the right figure in Fig. 6). TTS of QA is 0.20-0.54 ms, which is very fast even as the number of transactions increases. The TTS is almost twice as large as when r_m=0.99.

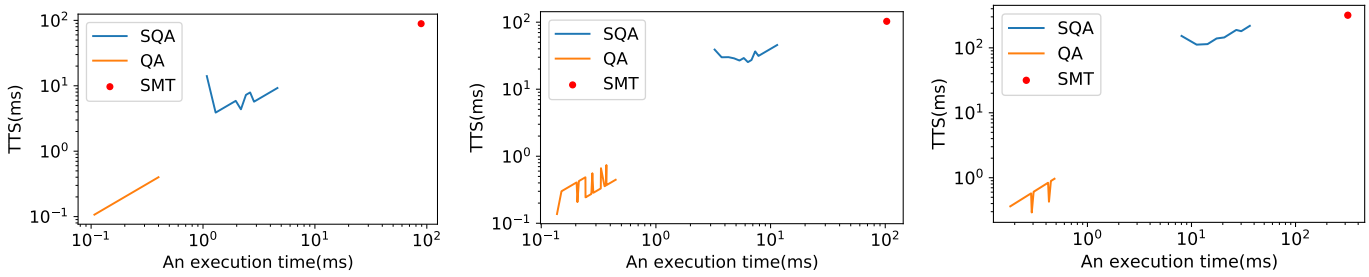


Fig. 5. Time to solution (TTS) with varying execution time

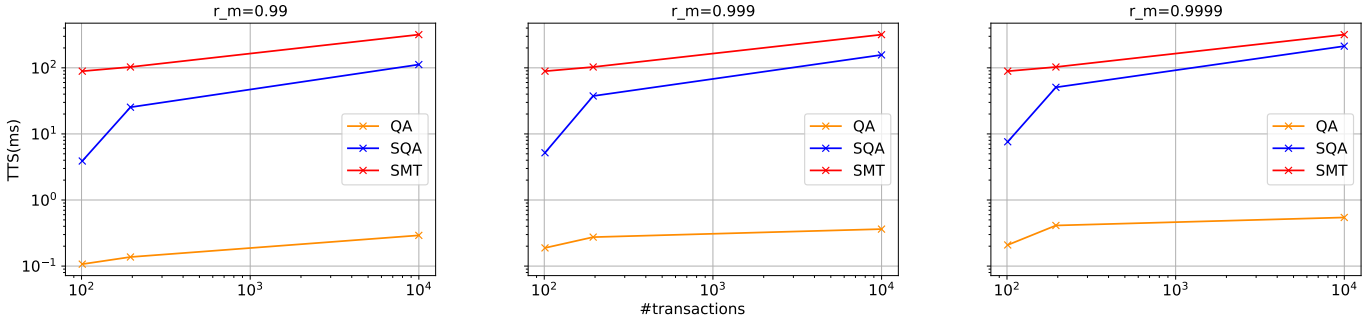


Fig. 6. Graph of change in TTS when scaling by the number of transactions

In other words, QA can increase the probability of success in obtaining a feasible solution by decreasing the execution speed by 2 times. The TTS of SQA is 7.6-213 ms, and the rate of increase is high. This is due to the low probability of obtaining a feasible solution for SQA, as can be seen from the probability of obtaining a feasible solution in Fig. 4. This is because the lower the probability of obtaining a feasible solution, the more runs are required to reach the target value of r_m.

E. Discussion

1) Performance Analysis: QA achieves almost 100% feasible solutions for all workloads in Fig. 4. This is because the "black-box serializability problem" is a satisfiability problem but not a problem minimizing objective function, thus making it easy to find the minimum value. In other words, obtaining a feasible solution is equivalent to finding an exact solution to the problem, which is more accessible than other problems because the number of possible combinations of exact solutions is greater. Therefore, the probability of obtaining a sufficiently good solution is high even with a small `annealing_time`, and this is considered to be the reason for this result.

2) Reliability Analysis: To obtain the correct solution more reliably, it is possible to introduce a system in which the optimization process is repeated until it succeeds in obtaining a feasible solution. When QA is executed, client programs can set the number of annealing operations in `num_reads` in advance and submit queries to the quantum annealing machine. The client program can determine whether the solution obtained by executing quantum annealing violates the QUBO constraints. If the constraint is violated in all of the `num_reads` solutions obtained, the client can run quantum annealing again under the same conditions. By repeating this until at least one feasible solution is obtained, the client program will always obtain the correct solution. This scheme delays the execution time by the multiple times it is triggered. However, the probability r_m used to calculate TTS depends on the value of `num_reads`. Increasing this value will cause the TTS to be larger, thus requiring more computation time for the usual execution. But it will also reduce the probability of a delay by 2 or more times in the event of failure to obtain a feasible solution. On the other hand, decreasing the value of `num_reads` increases the computation time of a simple execution, but increases the probability of a delay of more than twice as long. This tuning can be changed flexibly according to the requirements of the service using this method.

3) Scaling Analysis: Current QA implementations are limited by physical constraints such as the number of qubits and their topology. D-Wave Advantage 5.4, used in this experiment, has about 5,000 qubits, but the topology between qubits is a Pegasus graph, not a fully coupled graph. Therefore, when applying the problem to a QA machine, it is necessary to perform an operation called "minor embedding" so that multiple qubits are treated together as a single qubit. This operation lowers the number of variables to apply to the actual problem from the actual number of qubits. In the problem of

this paper, QUBO is relatively simple with only one constraint term; thus, the structure of the QUBO graph is feasible for larger scales at higher speeds.

Ising machines [18], [22] is designed to be used like QA to solve combinatorial optimization problems at high speed using simulators of QA and simulated annealing on GPUs, FPGAs, and other devices. These can be applied to larger problems faster than simulated annealing, although they might not be as fast as quantum annealing.

VI. Related Work

Metaheuristics [12] have been expected to be a powerful method to solve combinatorial optimization problems and continuously. Metaheuristics represent a higher level of heuristics, and it is expected to be able to obtain the optimal solution in a short time, even for very large problem sizes. Some examples are tabu search (TS) [14] [13], simulated annealing (SA) [16], and QA is one of them.

There are not many examples of QA being applied in the real world, except for the following. Florian et al. [17] used QA to optimize traffic flow in Beijing. Tobias et al. [19] realized air traffic control using QA. Sheir et al. [24] used QA to solve the multi-car paint shop problem, which optimizes the number of color switches between cars in a paint shop queue during manufacturing. They then compare the five classical methods with three quantum annealers. Ikeda et al. [15] formulated the nurse scheduling problem in QUBO, obtained the solution using QA, and evaluated it. Arindam et al. [11] formulated not only the nurse scheduling problem but also the physician scheduling problem and the nurse-physician scheduling problem assuming COVID-19 in QUBO. The quality of the solutions solved by QA is compared with that of SA. However, neither of the two researches applying QA to nurse scheduling problems compares the accuracy of the solution with other methods said to be effective for combinatorial optimization problems. They also do not compare the execution time of QA with that of other methods.

VII. Conclusion

The aim of this study was to speed up solving the "black-box serializability problem," which is difficult to solve on a classical computer due to its NP-completeness, by applying it to quantum annealing. The challenge was that as the number of transactions in history increased, the search area expanded exponentially.

The contribution of this paper is the formulation by QUBO to apply quantum annealing to the black-box serializability problem. We also compared a quantum annealing and simulated quantum annealing with the SMT solver used in Cobra, an existing method, and achieved a 751-890x speedup in terms of TTS, which takes into account the probability of obtaining a feasible solution.

We showed that quantum annealing can solve black-box serializability problems at extremely high speeds, but its current implementation is limited in the scale of problems it can solve. Its upper limit in this paper was around 10,000 transactions. Waiting for hardware evolution is one way of scaling it up, but another option is to use an Ising machine as a faster QUBO-based solver with classical accelerators such as GPU or FPGA than SQA on usual classical computers. Since Ising machine advances are active as well as quantum annealing, methods using the Ising model and QUBO would have a lot of potential for growth.

References

[1] AmazonAurora. https://aws.amazon.com/jp/rds/aurora/.

[2] AmazonDynamoDB. https://aws.amazon.com/jp/dynamodb/.

[3] Azure Cosmos DB. https://azure.microsoft.com/ja-jp/products/cosmos-db.

[4] Big Data in Real Time at Twitter. https://www.infoq.com/presentations/Big-Data-in-Real-Time-at-Twitter/.

[5] D-Wave Systems. https://www.dwavesys.com/.

[6] DBCobra. https://github.com/DBCobra.

[7] OpenJij. https://www.openjij.org/.

[8] Francisco Barahona. On the computational complexity of Ising spin glass models. *Journal of Physics A: Mathematical and General*, 15(10):3241–3253, 1982.

[9] Sam Bayless, Noah Bayless, Holger Hoos, and Alan Hu. Sat modulo monotonic theories. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 29, 2015.

[10] Philip A. Bernstein and Nathan Goodman. Multiversion concurrency control – theory and algorithms. *ACM Transactions on Database Systems (TODS)*, 8(4):465–483, 1983.

[11] Kunal Das, Sahil Zaman, Arindam Sadhu, Asmita Banerjee, and Faisal Shah Khan. Quantum annealing for solving a nurse-physician scheduling problem in covid-19 clinics, 2020.

[12] Tansel Dokeroglua, Ender Sevincb, Tayfun Kucukyilmaza, and Ahmet Cosar. A survey on new generation metaheuristic algorithms. *Computers & Industrial Engineering*, 137:106040, 2019.

[13] Fred Glover. Tabu search—part ii. *ORSA Journal on computing*, 2(1):4–32, 1990.

[14] Fred Grover. Tabu search—part i. *ORSA Journal on computing*, 1(3):190–206, 1989.

[15] Kazuki Ikeda, Yuma Nakamura, and Travis S. Humble. Application of quantum annealing to nurse scheduling problem. *Scientific reports*, 9(1):1–10, 2019.

[16] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *science*, 220(4598):671–680, 1983.

[17] Florian Neukart, Gabriele Compostella, Christian Seidel, David von Dollen, Sheir Yarkoni, and Bob Parney. Traffic flow optimization using a quantum annealer. *Frontiers in ICT*, 4:29, 2017.

[18] Takuya Okuyama, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Masanao Yamaoka. Binary optimization by momentum annealing. *Physical Review E*, 100(1):012111, 2019.

[19] Tobias Stollenwerk, Bryan O' Gorman, Davide Venturelli, Salvatore Mandra, Olga Rodionova, Hokkwan Ng, Banavar Sridhar, Eleanor Gilbert Rieffel, and Rupak Biswas. Quantum annealing applied to de-conflicting optimal trajectories for air traffic management. *IEEE transactions on intelligent transportation systems*, 21(1):285–297, 2019.

[20] Kadowaki Tadashi and Nishimori Hidetoshi. Quantum annealing in the transverse Ising model. *Physical Review E*, 58(5):5355–5363, 1998.

[21] Cheng Tan, Changgeng Zhao, Shuai Mu, and Michael Walfish. Cobra: Making transactional key-value stores verifiably serializable. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pages 63–80, 2020.

[22] Sanroku Tsukamoto, Motomu Takatsu, Satoshi Matsubara, and Hirotaka Tamura. An accelerator architecture for combinatorial optimization problems. *Fujitsu Sci. Tech. J*, 53(5):8–13, 2017.

[23] Gerhard Weikum and Gottfried Vossen. *Transactional Information Systems: Theory, Algorithms, and the Practice of Concurrency Control and Recovery*. Elsevier, 2001.

[24] Sheir Yarkoni, Alex Alekseyenko, Michael Streif, David Von Dollen, Florian Neukart, and Thomas Bäck. Multi-car paint shop optimization with quantum annealing. In *IEEE International Conference on Quantum Computing and Engineering (QCE)*, pages 35–41. IEEE, 2021.