SN Computer Science (2023) 4:609
https://doi.org/10.1007/s42979-023-02071-x

ORIGINAL RESEARCH

Optimizing the Production of Test Vehicles Using Hybrid Constrained Quantum Annealing

Adam Glos[1] · Akash Kundu[1,2] · Özlem Salehi[1]

Received: 22 August 2022 / Accepted: 21 June 2023

Abstract
Optimization of pre-production vehicle configurations is one of the challenges in the automotive industry. Given a list of tests requiring cars with certain features, it is desirable to find the minimum number of cars that cover the tests and obey the configuration rules. In this paper, we model the problem in the framework of satisfiability and solve it utilizing the newly introduced hybrid constrained quadratic model (CQM) solver provided by D-Wave. The problem definition is based on the "Optimizing the Production of Test Vehicles" use-case given in the BMW quantum computing challenge. We formulate a constrained quadratic model for the problem and use a greedy algorithm to configure the cars. We benchmark the results obtained from the CQM solver with the results from the classical solvers like coin-or branch and cut and Gurobi solver. We conclude that the performance of the CQM solver is comparable to the classical solvers in optimizing the number of test vehicles, given the noise-prone quantum hardware. However, the CQM solver takes much more time, which prohibits obtaining useful quantum advantages. As an extension to the problem, we describe how the scheduling of the tests can be incorporated into the model.

Keywords Vehicle configuration · Constrained quadratic model · Quantum annealing · D-Wave · BMW challenge

Introduction

Quantum computers are deemed promising technologies for solving industrial problems from various sectors like automotive, chemical, insurance, and technology. One of the main problem domains for industrial problems is optimization, as identified in the report prepared by Quantum Technology and Application Consortium (QUTAC) [1]. Recently, there have been attempts to solve optimization problems using near-term quantum computers, through variational quantum eigensolver (VQE) [2], quantum approximate optimization algorithm (QAOA) [3], and quantum annealing (QA) [4].

Quantum annealing is a heuristic method for solving optimization problems. It operates in the framework of quantum adiabatic computing, which is a quantum computing model alternative to gate based. Since many optimization problems are proven to be NP-hard, quantum annealing has gained significant interest as an up-and-coming tool to target them. Quantum annealers are commercially available by the D-Wave company [5], and a vast amount of research has been devoted to identifying potential use-cases [6]. D-Wave quantum annealers have been utilized to solve problems from different domains such as transportation [7–10], finance [11, 12], chemistry [13–15], and computer science [16–18].

Identified among the use-cases of quantum computing by BMW Group [19], optimization of pre-production vehicle configurations is one of the challenges in the automotive industry. Every year, new features and car components are launched by the companies, and various tests should be carried out before the series production. The tests under consideration range from the validation of the model's functionality to the evaluation of the new components. Consequently, pre-production vehicles are built for testing purposes. As the construction of pre-production vehicles is costly and complex [20], it is desirable to reduce the number of required test vehicles. Hence, the test cars should be configured to cover as many tests as possible, while meeting some dependency constraints among the different features.

✉ Akash Kundu
akundu@iitis.pl

1 Institute of Theoretical and Applied Informatics, Polish Academy of Sciences, Bałtycka 5, 44-100 Gliwice, Poland

2 Joint Doctoral School, Silesian University of Technology, Akademicka 2A, 44-100 Gliwice, Poland

Published online: 11 August 2023

Some of the attempts in solving the test-vehicle configuration optimization problem use the framework of satisfiability. The features of the vehicle are represented by Boolean variables, indicating whether the feature or the component exists or not. As each vehicle configuration should satisfy the feasibility rules concerning the different features of the car and the requirements imposed by the tests, such rules can be modeled through Boolean constraints. In Ref. [20], the authors present a Max-SAT framework that uses a greedy approach and tests it on small-scale real-world data. The problem is also studied by finding the minimum set cover in Ref. [21], where the authors formulate the problem as a minimum set cover problem and use SAT solver to check the feasibility of the configurations.

In this paper, our main goal is to exploit quantum annealing to solve the test-vehicle configuration problem. Our problem definition is based on the use-case “Optimizing the Production of Test Vehicles” given in the BMW quantum computing challenge [22] and takes into account various buildability constraints. We use an optimization approach, where all the variables are Boolean, and the conditions are given through Boolean constraints, and we aim to minimize the number of vehicles that will be used in testing. Among the various solvers provided by D-Wave, we use the newly introduced hybrid solver for constrained quadratic models [23]. The mentioned hybrid solver requires the problem to be encoded as a constrained quadratic model (CQM). In CQM, which is also known as the quadratically constrained quadratic programming in the literature, the problem is identified through a quadratic objective function and quadratic constraints defined over binary and integer variables. Once the problem is formulated, the hybrid solver takes advantage of both the classical heuristic methods and the D-Wave quantum processors. As the hybrid solver is proprietary to the D-Wave company, the exact way it operates is not revealed.

While modeling the problem as a CQM, our primary concern is to use as few qubits as possible. We propose an optimization model for which the total number of qubits grows in the order $O(n(f+o+q))$, where n, f, o, q are the numbers of vehicles, features, vehicle types, and tests, respectively, and the number of required qubits is independent of the number of constraints. The analysis applies for both decision and optimization formulations. The former answers the question “Are n cars sufficient to cover all the tests?”, and the latter aims at ‘maximize the number of tests covered using n test vehicles‘. To benchmark the results obtained from the CQM solver, we develop an integer linear programming formulation. Since both problems are notably time-consuming for current quantum and classical solvers, to benchmark the efficiency of the classical and quantum solvers, we analyze the performance of a greedy optimization procedure based on the optimization formulation. We test the performance of the algorithms on the dataset provided by the BMW quantum computing challenge [22] using D-Wave’s hybrid CQM solver, and the classical solvers CBC (coin-or branch and cut) and Gurobi solver. The results indicate that the performance of D-Wave’s hybrid solver in minimizing the number of required cars is comparable to the performance of the classical optimization algorithm. Furthermore, we consider a variation of the decision formulation that includes the scheduling of the tests. However, this model requires far more qubits; thus, we claim that it is particularly inefficient for practical purposes. As per the authors’ knowledge, this is the first attempt to benchmark the performance of the hybrid solver for CQM compared to classical solvers.

The rest of the paper is organized as follows. In “Preliminaries”, we present basic concepts related to SAT, Max-SAT, linear programming, quantum annealing, and constrained quadratic model. In “Buildability Constraints”, we describe the problem formulation and the optimization approach for solving the problem. In “Problem Solution”, we present and discuss the experimental results. We conclude with final comments in “Conclusion and Future Work”.

Preliminaries

In this section, we briefly explain the necessary background information on satisfiability problems, linear programming, and quantum annealing concepts.

Satisfiability Problems

Satisfiability problem [24] (SAT) is the problem of determining whether there exists an assignment to the binary variables that make a given Boolean expression true. A *Boolean expression* consists of *Boolean variables* $x_1, x_2, \dots, x_n$, that are combined together using logical OR ($\vee$) and AND ($\wedge$) operations and the negation operator ($\neg$). For instance, the Booelan expression $\phi = (\neg x_1 \wedge x_2) \vee x_3$ is satisfiable as $x_1 = 0, x_2 = 1, x_3 = 0$ is a satisfying assignment for ϕ. SAT is the first problem to be proven to be NP-complete [25, 26]. Hence, solving a large family of problems recognized as core to several areas in computer science and mathematics is as hard as solving the SAT problem.

A formula is said to be in *conjunctive normal form* (CNF), if it is written as the conjunction of clauses. A *clause* is a disjunction of *literals*, where a literal is either a variable (positive literal) or its negation (negative literal). Any Boolean formula can be expressed in CNF. For instance, we can express ϕ in CNF as $(\neg x_1 \vee x_3) \wedge (x_2 \vee x_3)$. SAT problem can be equivalently defined as the question of whether there exists an assignment to the Boolean variables that make the formula

$$C_1 \land C_2 \land C_3 \land \cdots \land C_m = \bigwedge_{i=1}^{m} C_i, \quad (1)$$

satisfiable, where C_i is a clause and there are m clauses in total.

Maximum satisfiability problem (Max-SAT) is a generalization of the SAT problem and can be considered as the optimization variant of the decision formulation. The goal is to find an assignment that maximizes the number of satisfied clauses. Note that knowing the optimal number of satisfied clauses, we can also deduce the solution to the SAT problem; therefore, we can conclude that the Max-SAT problem is NP-Hard.

A further generalization is the *weighted maximum satisfiability problem* (weighted Max-SAT), in which each clause is associated with a weight, and the aim is to maximize the weighted sum of the satisfied clauses.

Linear Programming

Linear programming (LP) is concerned with the optimization of an objective function subject to equality and inequality constraints, such that the objective and the corresponding constraints are linear. Linear programs can be expressed in the canonical form as

$$\begin{array}{ll} \text{minimize} & c^{\intercal}x \\ \text{subject to} & Ax \le b \\ & x \ge 0, \end{array}$$

where $c \in \mathbb{R}^n$, $A \in \mathbb{R}^{n \times n}$ and $b \in \mathbb{R}^n$. The aim is to find the vector of variables x that minimizes the objective function subject to the given constraints. When all variables x_i are integers, then the problem takes the name *integer linear programming* (ILP) which is NP-Hard in general. If the variables are further restricted to the set $\{0, 1\}$, then the problem is called *0–1 linear programming* (0–1 LP). Any ILP can be converted into 0–1 LP. For solving ILPs, there are heuristic methods like simulated annealing [27] and exact methods including cutting plane [28] and branch-and-bound methods [29].

There are various commercial and open-source solvers and toolkits for solving linear programs. PulP is a Python library [30] that provides tools for modeling problems and an interface for accessing various solvers. In this paper, we use PulP to model our problems and two different solvers to get the results: coin-or branch and cut (CBC) [31] solver, which is the default one in PulP, and Gurobi solver [32].

Quantum Annealing

Adiabatic quantum computing (AQC) is an analog computational model that relies on the quantum adiabatic theorem which states that a quantum state that is initially in the ground state is likely to stay in the ground state given that the evolution takes place slow enough. Some assumptions of AQC are lifted in *Quantum annealing* (QA), which is a meta-heuristic method for solving optimization problems using the quantum adiabatic theorem.

The problem Hamiltonian H_p is designed so that its ground state encodes the solution to the problem of interest, and an initial Hamiltonian H_0 is picked whose ground state is known and easy to prepare. The system is initialized with the ground state of H_0, and an adiabatic evolution path is followed so that the system ends up in the ground state of H_p. This is achieved by evolving the system with the time-varying Hamiltonian $H(t)$ expressed as

$$H(t) = \left(1 - \frac{t}{\tau}\right)H_0 + \frac{t}{\tau}H_p. \quad (2)$$

The quantum adiabatic theorem assures that the system always remains at the ground state of $H(t)$ for sufficiently large real evolution time τ. When $t = \tau$, $H(t)$ equals H_p and ideally the system is expected to be in the ground state of H_p.

Commercially available quantum annealers are provided by the D-Wave company. D-Wave quantum processing units (QPUs) implement the initial Hamiltonian $H_0 = -\sum_i \sigma_i^x$, where σ_i^x is the Pauli-X operator acting on i-th qubit and the problem Hamiltonian should be stated in the form of an Ising model $H_p = \sum_{i>j} J_{ij}\sigma_i^z\sigma_j^z + \sum_i h_i\sigma_i^z$, where J_{ij} denotes the interaction between sites i and j, h_i is the external magnetic field applied on-site i and σ_i^z is the Pauli-Z operator. Nevertheless, it is much more convenient to formulate problems over binary variables. Any problem that is expressed as a quadratic unconstrained binary optimization (QUBO) problem can be easily converted into an Ising model by replacing the binary variables b_i with $(1 - s_i)/2$. QUBO involves the minimization of a quadratic objective function defined over binary variables. While it is an unconstrained model, using the *penalty method*, one can incorporate the constraints to the objective function [33]. We would like to point out that any ILP formulation can be formulated as a QUBO and we refer readers to [8] for a detailed explanation.

When running a problem on D-Wave QPUs, the variables need to be mapped to the QPU architecture as the underlying graph representing the interactions in the QPU is not fully connected; this process is known as the minor embedding [34]. Hence, the number of variables in the QUBO formulation should be much smaller than the actual number of physical qubits, making it unachievable to run real-world problems because D-Wave Advantage QPU has 5640 qubits.

D-Wave hybrid solvers can solve much larger problems using a classical-quantum hybrid workflow. With the announcement of the new hybrid solver for constrained quadratic models (CQMs), D-Wave's hybrid solver service (HSS) now consists of three different solvers. The binary quadratic model (BQM) solver accepts problems defined in the form of QUBO. One can define problems over discrete variables in an unconstrained form and use the discrete quadratic model (DQM) solver. In this paper, we will use the CQM solver, which is described in more detail in "Constrained Quadratic Model". All solvers in HSS follow the same workflow. After taking the input, classical heuristic solvers that run parallel on the cloud are called. These solvers have heuristic and quantum modules that send queries to D-Wave Advantage QPU. The responses taken from the QPU are used to guide the classical heuristic process and improve the quality of the solutions obtained so far. Finally, the heuristic solver returns a solution to the user.

Constrained Quadratic Model

Constrained quadratic model (CQM) is the name given by D-Wave to the model involving a quadratic objective function and quadratic constraints that are defined over binary or integer variables. In the literature, this is also known as *quadratically constrained quadratic programming*. We can define a CQM as

$$\begin{array}{lrl} \text{minimize} & & x^\intercal P_0 x + q_0^\intercal x \\ \text{subject to} & x^\intercal P_i x + q_i^\intercal x \le r_i & i = 1, \dots, m \\ & & x \ge 0, \end{array}$$

where $P_i \in \mathbb{R}^{n \times n}$, $q_i \in \mathbb{R}^n$ and $r_i \in \mathbb{R}$ for $i = 0, 1, \dots, m$. The goal is to find the vector x that minimizes the objective function, where x consists of binary and integer variables.

D-Wave has recently introduced the hybrid solver for CQM. Unlike the previous hybrid solvers and quantum annealers of D-Wave, the CQM solver naively supports equality and inequality constraints. This is advantageous for problems involving constraints compared to the QUBO, which is the standard formulation that has been used for quantum annealing so far. First of all, there is no need for removing the constraints through the penalty method, which in turn increases the number of variables if the constraints are in the form of inequalities. Second, it removes the difficulty of setting penalty coefficients, which is challenging as the model becomes sophisticated. Third, CQM solver allows the inclusion of quadratic constraints directly into the model, which is not possible for QUBO. It is also suggested by D-Wave that the hybrid CQM solver should be preferred over the other hybrid solvers in case the problem naturally involves constraints. An experimental evidence for performance comparison is available in Ref. [23].

Formulation of the Problem

In this section, we will describe the details of the vehicle optimization problem and model it through binary variables and linear constraints. We provide an integer linear program which can be directly used with the classical solvers and D-Wave CQM solver. We would like to note that the integer linear program is novel and has not been proposed before.

Overview

The configuration of each vehicle is determined by the presence or absence of the availability of the features, and each vehicle has a specific type. Let us discuss the buildability constraints that a vehicle configuration should satisfy [22].

- **Single type requirement:** Each vehicle should have a single type.
- **Features allowed per type:** For a given type, only some of the features are available.
- **Group features:** Certain groups of features cannot be implemented together (i.e., at most one can be implemented). This constraint is valid for all types.
- **Rules per type:** For each type, there are rules which govern the feature set that the vehicle of a specific type should have. Those are mainly implication rules.

Finally, we have the **test requirements** that define the properties of the cars needed for the testing phase. We will assume that each test requires a single car and discuss how this assumption can be removed later on.

Consider n test vehicles, a list of f features and assume that there are o different types available. Let us assume that there are q test requirements. For vehicle $i \in [n]$, we will represent the presence/absence of feature $j \in [f]$ through the binary variables $b_{i,j}$, where $b_{i,j} = 1$ if and only if (iff) vehicle i has feature j. Next, we represent the type of the vehicle using the variable $t_{i,j}$, where $t_{i,j} = 1$ iff vehicle $i \in [n]$ is of type $j \in [o]$. And finally, we define the binary variables $p_{i,j}$, to represent whether a vehicle is used in a test, where $p_{i,j} = 1$ iff vehicle $i \in [n]$ is used for test $j \in [q]$.

Suppose that there are c buildability constraints in total. Note that the same set of constraints applies to each vehicle. Each constraint ϕ_k can be expressed using a Boolean expression over the binary variables we have defined above. First of all, we would like all buildability constraints to be satisfied for any given number of cars. This can be expressed using the following logical expression:

$$\bigwedge_{i=1}^{n} \bigwedge_{k=1}^{c} \phi_k(b_{i,1}, \dots, b_{i,f}, t_{i,1}, \dots, t_{i,o}). \tag{3}$$

Similarly, we can express the test requirements using Boolean expressions. To start with, each test requires absence or presence of certain features. We need constraint ψ_l to ensure that the variable $p_{i,l}$ representing the l'th test requirement is set correctly:

$$\bigwedge_{i=1}^{n} \bigwedge_{l=1}^{q} \psi_l(b_{i,1}, \dots, b_{i,f}, p_{i,l}). \tag{4}$$

Note that if $p_{i,l} = 0$, then the expression $\psi_l(b_{i,1}, \dots, b_{i,f}, p_{i,l})$ always evaluates to 1. This is because if car i is not used in test l there is no need to verify constraint ψ_l. Now, we can identify two different problems based on how we interpret test requirements. Given n vehicles, the first problem is to decide whether there exist configurations for the given cars so that the buildability constraints are satisfied, and for each test requirement, then there is at least one car satisfying the requirement. This results in the following *satisfiability problem*:

$$\begin{aligned} &\bigwedge_{i=1}^{n} \bigwedge_{k=1}^{c} \phi_k(b_{i,1}, \dots, b_{i,f}, t_{i,1}, \dots, t_{i,o}) \\ &\wedge \bigwedge_{i=1}^{n} \bigwedge_{l=1}^{q} \psi_l(b_{i,1}, \dots, b_{i,f}, p_{i,l}) \wedge \bigwedge_{l=1}^{q} \bigvee_{i=1}^{n} p_{i,l}. \end{aligned} \tag{5}$$

If one wants to find the smallest number of vehicles for which the Boolean formula given in Eq. (5) is true, then the bisection method can be used by starting with a large n and applying binary search to find the optimal value.

It can be the case that the number of vehicles is fixed, and the aim is to find a configuration of vehicles that satisfies the buildability constraints and maximize the number of satisfied test requirements. So, unlike in the case of SAT, it is not required that all the test requirements are satisfied. Furthermore, each test requirement can be assigned some weight, in which case the aim is to maximize the weighted sum of the fulfilled tests. This yields a variant of *weighted maximum satisfiability problem* that is expressed mathematically as follows:

$$\text{maximize } \sum_{l=1}^{q} w_l \prod_{i=1}^{n} p_{i,l}, \tag{6}$$

where w_l is the weight associated with test requirement l, subject to the constraints

$$\bigwedge_{i=1}^{n} \bigwedge_{k=1}^{c} \phi_k(b_{i,1}, \dots, b_{i,f}, t_{i,1}, \dots, t_{i,o}) \wedge \bigwedge_{i=1}^{n} \bigwedge_{l=1}^{q} \psi_l(b_{i,1}, \dots, b_{i,f}, p_{i,l}). \tag{7}$$

Note that this problem is not originally given as an SAT or Max-SAT instance—instead, it was defined through the collection of rules to be followed [22]. Instead of formulating an SAT or Max-SAT problem and then transforming it into ILP, we directly construct an ILP which turned out to be simpler and more efficient. Since such a formulation is a special case of CQM solver, it can be directly solved by the the solver provided by D-Wave.

Buildability Constraints

Having discussed the general overview of the vehicle testing problem, now we are ready to express it as a linear program defined over binary variables. We will be expressing Boolean expressions that correspond to constraints using equalities and inequalities. We will start with the buildability constraints, and we will either provide inequality or equality constraints in each case.

Single Type Requirement

The fact that each vehicle should have a single type can be incorporated using the constraints

$$\sum_{j=1}^{o} t_{i,j} = 1, \quad i \in [n]. \tag{8}$$

Features Allowed Per Type

Some of the features are not allowed for a specific type. We encode this constraint through the features which are not allowed for the given type. In other words, for each type $j \in [o]$, there exists a collection of features F_j such that $t_{i,j} = 1 \implies b_{i,k} = 0$ for all $k \in F_j$ for all vehicles $i \in [n]$. This is expressed by the inequality constraints

$$t_{i,j} + b_{i,k} \le 1, \quad i \in [n], \quad j \in [o], k \in F_j. \tag{9}$$

Group Features

Let F_G denote a collection of feature groups. Given a group of features $G \in F_G$, we need to make sure that at most one feature from each group G is implemented. This is equivalent to inequalities

$$\sum_{k \in G} b_{i,k} \le 1, \quad i \in [n], \tag{10}$$

for each $G \in F_G$.

Rules Per Type

Depending on the type of the vehicle, one may define rules in the form of implications about the presence or absence of the features in the vehicle as

Table 1 Some logical expressions and their corresponding constraints

Type	Position	Logical expression	Corresponding constraint
m&	LHS	$t_{i,j} \wedge \bigwedge_{r=1}^{M} b_{i,j_r} \wedge \bigwedge_{r=1}^{\mu} b_{i,l_r}$	$1 - t_{i,j} + M - \sum_{r=1}^{M} b_{i,j_r} + \sum_{r=1}^{\mu} b_{i,l_r} = 0$
0&	RHS	$\bigwedge_{r=1}^{N} b_{i,k_r}$	$N - \sum_{r=1}^{N} b_{i,k_r} = 0$
1&	RHS	$\bigwedge_{r=1}^{N} \neg b_{i,k_r}$	$\sum_{r=1}^{N} b_{i,k_r} = 0$
0|	RHS	$\bigvee_{r=1}^{N} b_{i,k_r}$	$1 \leq \sum_{r=1}^{N} b_{i,k_r}$
1&	RHS	$\bigvee_{r=1}^{N} \neg b_{i,k_r}$	$\sum_{r=1}^{N} b_{i,k_r} \leq N - 1$

where M is the number of positive literals on the LHS, μ is the number of negative literals on the LHS, and N is the number of literals on the RHS

$$\texttt{T1} : \texttt{F2} \wedge \neg\texttt{F4} \wedge \neg\texttt{F5} \implies \texttt{F1} \vee \texttt{F3},$$

where `T1` is the type of the vehicle, and on the left and right sides of the implication, we have conjunction or disjunction of literals. The rule is saying that if a vehicle is of type 1, has feature 2, and does not have features 4 and 5, then it should have at least one of features 1 or 3.

We group the constraints into several classes depending on the form of the constraint and label it with a 4-character string, where the first two characters are representing the LHS of implication, and the remaining two are the RHS. The first character describes whether the variables on the LHS of the implication are negated or not. There are three possibilities: "0" if none of the variables are negated, "1" if all variables are negated, "m" if some of the variables are negated. Second character on LHS describes the operator used: "&" for AND, "|" for OR, 1 if only a single variable exists. The next two characters have the same meaning but describe the RHS of the constraint. For example, the implication above belongs to the class m&0|. The cases we will consider are inspired by the BMW quantum computing challenge dataset and include the combinations of m& and 0| on the LHS and 0|, 0&, 1|, 1& on the RHS. We give the logical expression and the corresponding arithmetic expression for the mentioned cases in Table 1.

Now, we will investigate the implications of each type, making use of the arithmetic expressions given in Table 1. Where, M is the number of positive literals on the LHS, μ is the number of negative literals on the LHS, and N is the number of literals on the RHS.

Case m&0& This category encapsulates m&01, 0&0&, 1&0&, 0&01, 1&01, 010&, 0101, 110&, 1101. The constraints take the form

$$N - \sum_{r=1}^{N} b_{i,k_r} \leq N\left(1 - t_{i,j} + M - \sum_{r=1}^{M} b_{i,j_r} + \sum_{r=1}^{\mu} b_{i,l_r}\right), \quad i \in [n]. \tag{11}$$

RHS is 0 iff the assumption is satisfied, in which case $\sum_{r=1}^{N} b_{i,k_r}$ should be equal to N. If the assumption is not satisfied, then RHS is at least N, hence the inequality is still correct.

Case m&1& This category encapsulates m&11, 0&1&, 1&1&, 0&11, 011&, 0111 and the reasoning is the same as above

$$\sum_{r=1}^{N} b_{i,k_r} \leq N\left(1 - t_{i,j} + M - \sum_{r=1}^{M} b_{i,j_r} + \sum_{r=1}^{\mu} b_{i,l_r}\right), \quad i \in [n]. \tag{12}$$

Case m&0| This category encapsulates 0&0|, 1&0|, 010|, 110|. The constraints take the form

$$1 - \sum_{r=1}^{N} b_{i,k_r} \leq \left(1 - t_{i,j} + M - \sum_{r=1}^{M} b_{i,j_r} + \sum_{r=1}^{\mu} b_{i,l_r}\right), \quad i \in [n]. \tag{13}$$

Note that RHS is 0 iff the assumption is satisfied In this case $\sum_{r=1}^{N} b_{i,k_r} \geq 1$ should be true. If the assumption is not satisfied, then the inequality is still correct.

Case m&1| This category encapsulates 0&1|, 1&1|, 011|, 111|. The constraints take the form

$$\sum_{r=1}^{N} b_{i,k_r} - N + 1 \leq \left(1 - t_{i,j} + M - \sum_{r=1}^{M} b_{i,j_r} + \sum_{r=1}^{\mu} b_{i,l_r}\right), \quad i \in [n]. \tag{14}$$

Case 0|0|, 0|1| We take the contra positives and obtain 1&1& and 0&1&, respectively, which are already discussed above.

Case 0|1&, 0|0& The constraints of the form 0|1& are expressed as $t_{i,j} \wedge \bigvee_{r=1}^{M} b_{i,j_r} \implies \bigwedge_{r=1}^{N} \neg b_{i,k_r}$ and equivalently, we have conditions

$$(t_{i,j} \wedge b_{i,j_1}) \implies \bigwedge_{r=1}^{N} \neg b_{i,k_r}, \tag{15}$$

$$\vdots$$

$$(t_{i,j} \wedge b_{i,j_M}) \implies \bigwedge_{r=1}^{N} \neg b_{i,k_r}. \tag{16}$$

We have M rules of the form `0&1&`. Similarly, the constraints of the form `0|0&` translate to rules of the form `0&0&`.

Test Requirements

Test requirements define the properties of cars needed for the testing phase. Each test requires the absence or presence of certain features. We will assume that the test j is in the form

$$b_{i,k_1} \wedge \cdots \wedge b_{i,k_{T_j^1}} \wedge \neg b_{i,l_1} \wedge \cdots \wedge \neg b_{i,l_{T_j^2}} \wedge (b_{i,m_1} \vee \cdots \vee b_{i,m_{T_j^3}}), \tag{17}$$

the values T_j^1, T_j^2 and T_j^3 may be equal to 0.

Recall that $p_{i,j}$ indicates whether vehicle i is used in test j. We need constraints to ensure that the binary variables $p_{i,j}$ are properly set. Note that test j either imposes some features to exist in the vehicle, in which case we can express it using the inequality

$$-b_{i,k_r} + p_{i,j} \le 0, \quad i \in [n], \quad r \in [T_j^1], \tag{18}$$

or imposes that some features should not exist in the vehicle, which results in the inequality

$$b_{i,l_r} + p_{i,j} \le 1, \quad i \in [n], \quad r \in [T_j^2], \tag{19}$$

or imposes disjunction of some features which translates as the inequality

$$p_{i,j} - \sum_{r=1}^{T_j^3} b_{i,m_r} \le 0, \quad i \in [n]. \tag{20}$$

We will call the constraints defined in 18–20 as the test constraints. The test constraints are needed both in decision and optimization approaches.

We will lift the assumption that each test requires only a single car. Let us assume that test j requires k_j cars that satisfy the required properties. In case we want to solve the decision problem, we include the following constraint in our formulation to ensure that the number of vehicles that satisfy test j is k_j for each $j = 1, \dots, q$:

$$\sum_{i=1}^{n} p_{i,j} = k_j, \quad j \in [q]. \tag{21}$$

If we want to solve the optimization problem, then we need to define an objective function to maximize. One possibility is to take into account the number of cars that satisfy the test and reflect this in the weight. This results in the following objective function:

$$\sum_{i=1}^{n} \sum_{j=1}^{q} w_j p_{i,j}. \tag{22}$$

There should be some upper bound on the number of cars that satisfy a specific test. For example, for a given test j, if more than k_j cars satisfy the test, the excess ones should not add to the objective. Hence, we need the following constraint in the case of the optimization approach:

$$\sum_{i=1}^{n} p_{i,j} \le k_j, \quad j \in [q]. \tag{23}$$

To conclude, to solve both the decision and optimization problem, we need the buildability and the test constraints defined in 8–20. For the decision problem, we need additionally the constraint defined in Eq. (21). For the optimization, we need additionally the constraint defined in Eq. (23) and the objective function is defined as in Eq. (22).

Scheduling

A related problem in the automotive industry is the scheduling of vehicle tests. Aside from the configuration of the vehicles, there are additional constraints regarding when and how the tests will be conducted and whether a vehicle may be used in more than one test, making the problem more complex. The problem has been considered using classical approaches like constraint programming and mixed-integer linear programming in Refs. [30, 31, 35]. From our perspective, the presented model can be extended to incorporate scheduling constraints, as we will discuss briefly.

We assume that each test takes a single day, and the tests should be completed in D days. We extend the previously introduced $p_{i,j}$ variables into $p_{i,j,d}$ where $d \in [D]$ is the day at which test j is performed with vehicle i. We replace $p_{i,j}$ in each previously introduced condition with $p_{i,j,d}$, and if needed a summation over d should be added. For example, constraint given in Eq. (21) will be replaced with

$$\sum_{d=1}^{D} \sum_{i=1}^{n} p_{i,j,d} = k_j, \tag{24}$$

for each test $j \in [q]$.

From now on, we assume that only a single car is needed for each test. If the j-th test requires k_j cars, we create variables $p_{ij_1}, p_{ij_2}, \dots, p_{ij_{k_j}}$ for test j. Hence, the constraint presented in Eq. (24) takes the form

$$\sum_{d=1}^{D} \sum_{i=1}^{n} p_{i,j,d} = 1, \tag{25}$$

where j belongs to the extended list of tests.

To ensure that test j is performed within the time frame $[t_j^{\text{start}}, t_j^{\text{end}}]$, we need the constraint

$$t_j^{\text{start}} \le d \cdot \sum_{d=1}^{D} \sum_{i=1}^{n} p_{i,j,d} \le t_j^{\text{end}}. \tag{26}$$

If for a given test set $\mathcal{J} = \{j_1, \dots, j_m\}$ we need to use different vehicles for testing, then the vehicle should be used at most once for one of the tests in the given test group. This constraint can be imposed by

$$\sum_{d=1}^{D} \sum_{j \in \mathcal{J}} p_{i,j,d} = 1, \tag{27}$$

for all $i \in [n]$ and each test set $\mathcal{J}$.

Let us now consider other conditions on scheduling. Let K be the number of cars that can be tested per day. We need to ensure that for each day $d \in [D]$, the number of tests performed is at most K, which is equivalent to

$$\sum_{i=1}^{n} \sum_{j=1}^{q} p_{i,j,d} \le K. \tag{28}$$

Similarly, one has to ensure that each car $i \in [n]$ is tested at most once in each day $d \in [D]$, which is equivalent to

$$\sum_{j=1}^{q} p_{i,j,d} \le 1. \tag{29}$$

Let us now consider how one can assign groups to each test to impose an order condition among tests from different groups. Let g_j be the group id of the test j. We assume, that g_j is an integer in $\{1, \dots, \bar{g}\}$, s.t. for two tests j, j', where j has to be performed before j' if $g_j > g_{j'}$ (*order condition*). In addition, we assume that the tests from group $g_j = 1$ are full crash tests, thus not only that they have to be the final test, but also each car can be used only once for such test (*crash condition*).

The order condition can be implemented as follows: For each vehicle $i \in [n]$, for each pair of tests j, j' such that $g_j > g_{j'}$, and for each $d, d' \in [D]$ such that $d < d'$, we add the constraint

$$p_{i,j',d} + p_{i,j,d'} \le 1. \tag{30}$$

Note that with the above approach, we can handle even more complicated test ordering, like the one defined by a partial order of tests.

Implementing the crash condition, together with the order condition, is enough to ensure that the car is used for only one crash test. Let $J_1 = \{j \in [q] : g_j = 1\}$ be the set of tests resulting in a crash. The constraint takes the form of each vehicle $i \in [n]$

$$\sum_{j \in J_1} \sum_{d=1}^{D} p_{i,j,d} \le 1. \tag{31}$$

Resource Analysis

Let us analyze the number of variables and constraints required by the formulation. To start with, there exist $n \cdot f$ binary variables $b_{i,j}$, $n \cdot o$ binary variables $t_{i,j}$, and $n \cdot q$ binary variables $p_{i,j}$. Overall, we need $O(n(f + o + q))$ binary variables, which grow linearly in the number of vehicles.

There are c buildability constraints. The number of test requirements depends on the individual tests and can be expressed as $q + \sum_{j=1}^{q} T_j^1 + T_j^2 + [T_j^3 > 0]$, where $[T_j^3 > 0] = 1$ if $T_j^3 > 0$. The first term results either from the constraint Eq. (21) or Eq. (23), depending on the problem in consideration. Assuming that T_j^1 and T_j^2 are negligible compared to q, the total number of constraints can be expressed as $O(c + q)$.

Let us now consider the number of qubits used for scheduling constraints. Previous considerations are still valid up to the part where $p_{i,j}$s were computed, as they are now replaced with $p_{i,j,d}$. So in total, we need $O(n(o + f + qD))$ variables. Note that the polynomial is no longer quadratic.

Problem Solution

In this section, we will describe our implementation details and present our results

Algorithm

Based on the formulations presented, one can follow different approaches to find the minimum number of required vehicles. The first is the global bisection method, which is also proposed in BMW use-case specification. The idea is to start with a large n value and then use binary search to find out the optimal n. As the number of variables grows as the product of the number of vehicles and the number of constraints, the limitation of this approach is a large number of variable requirements. For instance, in the case of the CQM solver, the number of variables is limited to 5000. In the case of Gurobi solver, there is no limit on the number of variables and constraints that can be used in principle, however, the time needed for solving the problem increases as the number of variables and constraints increase, which may result in an intractable problem in practice.

Another approach would be the vehicle-greedy algorithm. We choose an extra parameter n_{bunch} which denotes the number of cars that will be configured at each iteration. After each iteration, the tests are updated by removing the satisfied ones and by diminishing the number of required cars for a test if it is partially satisfied. Then the optimization process is repeated with the new n_{bunch} cars. The procedure stops

Table 2 A summary of the number of constraints based on the real-world problem

Constraint	Number
Features allowed per type	25
Rules per type	4032
Group features	41
Test requirements	643

after all the tests are covered. Note that we actually maximize the number of covered tests using this approach, thus solving the optimization problem instance.

Implementation

We used the dataset provided by the BMW quantum computing challenge, which is created based on the BMW Series 2 Gran Coupe. The features and constraints are based on the actual numbers resulting in a real-world problem. The specifications of the dataset are given in Table 2.

When analyzing the buildability constraints, we noticed a significant redundancy in the "rules per type" constraints. We realized that some of the constraints apply to all types. Second, some of the constraints apply to all types. In this case, we used a type-independent constraint and heavily reduced the number of constraints. For instance, assuming that the inequality $b_{i,1} \leq 2 - t_{i,j} - b_{i,2}$ exists for all $i \in [n]$ and $j \in [o]$, it can be replaced by $b_{i,1} \leq 1 - b_{i,2}$ for all $i \in [n]$. In case some constraint was missing for several types, but its inclusion for the remaining types was not disruptive (since the left-hand side of the implication could not be satisfied for the particular type with any combination of features), we assumed that it applies to all types and used the discussed simplification. When some features were not available for the given type yet appeared as a positive literal in the constraint, they were removed. Finally, some "rule per type" constraints possessed redundant information because variables were repeated both on the left and right sides. Those constraints were simplified as well, resulting in new constraint types, which are implemented in a form similar to the ones previously mentioned. Besides the buildability constraints, we also performed simplification for the test requirements, by merging the test lines occurring multiple times in the file. Further details can be found in our implementation processing which can be found in the [36].

We used the vehicle-greedy algorithm, taking $n_{\text{bunch}} = 1$, hence optimizing a single vehicle at a time. After the mentioned simplifications, the model has 911 binary variables and 6313 constraints. We followed the optimization approach taking Eq. (22) as the objective function and setting all weights equal to 1. We used `PuLP toolkit` and `dwave-ocean-sdk` to implement the code for generating the linear program and the constrained quadratic model. PuLP, Gurobi and CBC solvers were used with their default settings and no time limit was given. The experiments were run on a computer with the following specifications: Intel(R) Core(TM) i9-10900KF CPU @ 3.70 GHz; Ubuntu 20.04.3 LTS, 64 GB RAM.

Results and Discussion

As mentioned earlier, we obtained the results by setting $n_{\text{bunch}} = 1$, i.e., one car is configured at each iteration. Hence, the number of successful iterations (which outputs a valid car) corresponds to the number of needed vehicles. The algorithm stops if no tests are remaining to be satisfied. For the classical solvers, the experiments are repeated 60 times, and for CQM solver, it is repeated 3 times. CBC and Gurobi algorithms always return the same result in each experiment, i.e., they terminate at the same iteration and return the same number of cars. Meanwhile, for CQM solver, we took the best possible outcome.

In the Fig. 1a, we illustrate the number of remaining tests after each iteration using CQM, CBC, and Gurobi solvers setting $n_{\text{bunch}} = 1$. CBC solver returns the smallest number of vehicles which is 62, and Gurobi solver returns 64. We would like to note that in the experiment with CQM solver, one test that requires a single vehicle remains after the 65th iteration. Thus, we can conclude that the CQM solver returns 66; however, the solver fails to find a configuration that satisfies the test in the 66'th iteration.

The problem size gets smaller over the iterations as some tests are removed. We analyze how the runtime changes as the problem size gets smaller in Fig. 1b for $n_{\text{bunch}} = 1$. The runtime is calculated by taking the average over the repeated experiments. The primary observation while comparing the performance of CQM, CBC, and Gurobi is that after the first few iterations, the runtime of the CQM solver saturates near 5 s. This is because the default runtime for the solver is 5 s, and one can not go below it. Meanwhile, the fluctuation in runtime for CBC solver with the number of iterations is visible and varies in the range of 1–25 s, and the fluctuations in runtime comparatively stabilize for the number of iterations ≥ 30. Finally, for Gurobi solver, the runtime always stays ≤ 1 s; hence, it takes the least amount of time to satisfy all the tests. In Table 3, a summary of the overall runtime taken by the solvers is depicted. Overall, it takes 6.309 s for Gurobi solver to find the solution, which is significantly smaller compared to the other solvers. Although CBC solver takes a longer time, it returns the best solution. CQM solver performs worse both in runtime and minimizing the number of required cars, it takes 371.975 s, in which only 0.437 s spent on QPU.

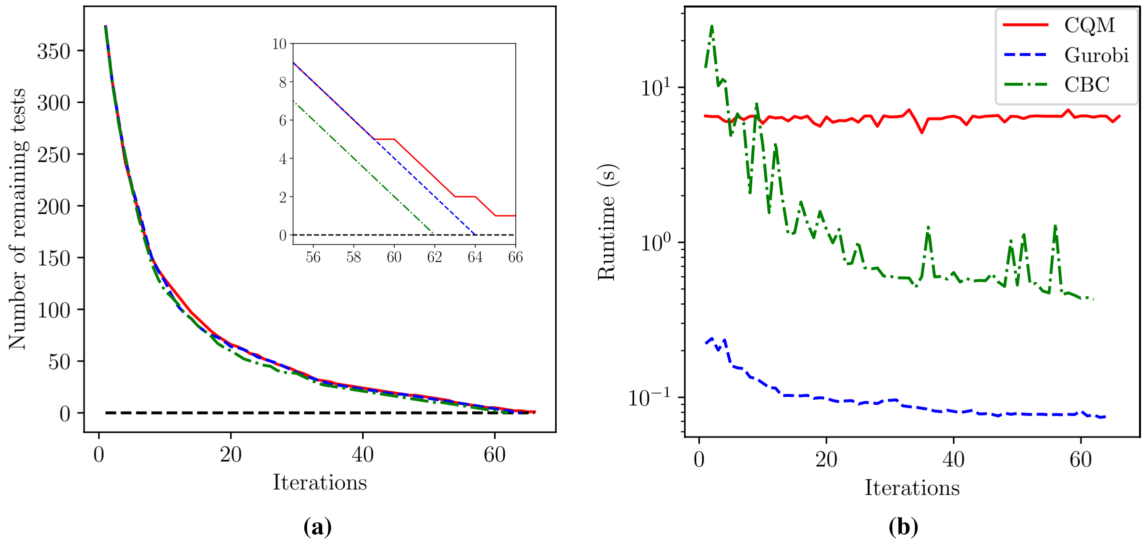

Fig. 1 Illustration of variation in **a** number of remaining tests and **b** runtime with respect to number of iterations for CQM, CBC and Gurobi solvers

Table 3 The averaged overall runtime by the CBC, Gurobi (classical) and CQM (quantum) solvers

Solver	Overall runtime (in seconds)
Gurobi	6.309
CBC	136.800
CQM	371.975

It seems evident that CBC and especially Gurobi solvers provide slightly better answers using significantly shorter time. However, in the very first iterations, we can see that the CQM solver is providing similar quality results in a slightly shorter time than CBC. Considering the shape of the time dependency observed in Fig. 1, it is reasonable to expect that there are problem instances for which the CQM solver outperforms CBC. Note that we cannot conclude that CQM solver will be better for arbitrarily large problem instances, as the time dependency cannot be observed due to fixed optimization time; hence, any potential outperformance of the CQM solver may be limited to a specific interval of problem instance sizes.

Considering this, we can make an attempt to estimate the minimum number of variables for which we might expect to obtain similar results for the Gurobi and CQM solver. Since it is anticipated that the time complexity of ILP optimization is exponential with the size of the data, we analyzed the linear dependency between the logarithm of time and the number of variables, which is depicted in Fig. 2. Here, we focused on the runtime per iteration and its corresponding number of variables in that iteration, rather than the overall

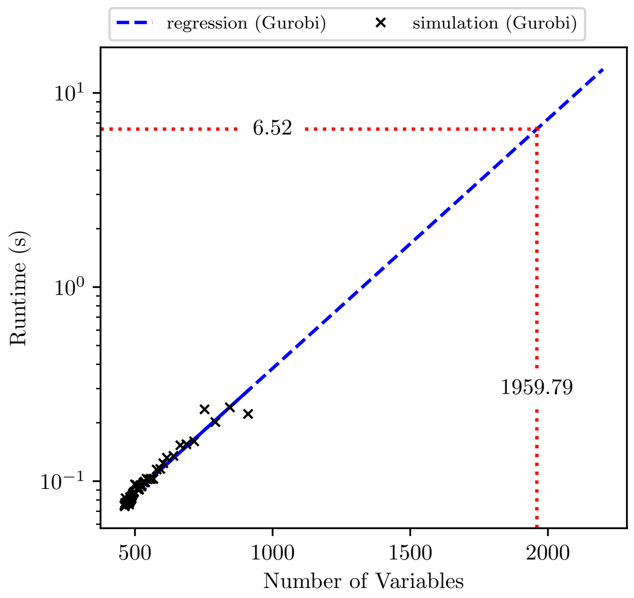

Fig. 2 The dependency between the number of variables for each iteration vs. runtime of the given iteration. The blue dashed line is the regression line of log(runtime) $= A \times$ (number of variables) $+ B$. The horizontal red dotted line is the average time over all iterations for CQM solver, and the vertical red dotted line is the corresponding number of variables for Gurobi solver based on the regression line. Note that the crossing point is far away from the data we used for regression

runtime across all iterations. Additionally, we also included the average runtime required by the CQM solver for comparison. Through this analysis, we can estimate the minimum number of variables that could potentially benefit from the

CQM solver. Note that if the CQM solver starts to require more optimization time within this range, the number of variables for which we would observe benefits may increase.

We have also checked the performance of the solvers when $n_{\text{bunch}} = 5$, which is the maximum possible number that can be taken without exceeding the 5000 variables[1] limit of the CQM solver. Gurobi solver returned 63 cars, so we can say that the result is slightly improved. However, the overall experiment took a significantly longer time (83 s). For the CQM solver, no feasible solutions were obtained with the default time limit of 5 s. We observed that the returned samples either violated the "single type constraint" and the vehicles had no type (in that case all constraints related to "rules per type" are automatically satisfied) or several other constraints were violated. When the time limit was increased to 10 s, then the CQM solver was able to return a feasible solution at each iteration. However, after the 13th iteration (after 65 vehicles were configured), there were still 35 tests remaining to be satisfied, hence the optimization quality was worse. We would like to remark that the CBC solver failed to return any result within a reasonable amount of time.

Conclusion and Future Work

In this paper, we proposed a greedy algorithm for solving the optimization problem of the production of test vehicles using the new hybrid CQM solver by D-Wave. We provided a constrained quadratic model formulation for the problem that requires the number of qubits linearly proportional to the number of vehicles, car types, features, and tests. We implemented the code for generating the constrained quadratic model and solved the problem instance provided by BMW quantum computing challenge on D-Wave CQM solver. We benchmarked the results by implementing the integer linear program formulation and running the same algorithm using classical solvers like CBC and Gurobi.

The results show that CQM solver gives comparable quality results to classical solvers in optimizing the number of required vehicles. However, the classical solvers require much less time to provide slightly better results. Keeping in mind the challenges faced and the ongoing efforts in the development of quantum computers, CQM solver has the potential to be a promising tool for large problems in the near future, provided that the algorithm will be significantly sped up. Note that the solver relies on quantum hardware, which is an advanced, very recent noise-prone technology.

The current CQM Solver is limited to 500,000 variables, while the real-world problems which are not tractable for classical solvers often require more than that. The problem size is an important factor as the currently available quantum solvers are limited in the number of qubits. Several works try to formulate models for gate-based quantum computers that are more efficient in the number of qubits used [37–41] and further research can be pursued in this direction for the considered problem.

A related and more general problem is product configuration and reconfiguration, where the problem's scope is not restricted to vehicles and one may consider any product such as computer parts. Satisfiability-based approaches have been considered in Refs. [42, 43]. The presented model can be extended for such problems and evoke potential use-cases for D-Wave CQM solver.

[1] At the time of conducting the experiments, the CQM solver was limited to 5000 variables.

Acknowledgements We would like to thank Jarosław Miszczak, Krzysztof Domino, and Aleksandra Krawiec for discussion on the subject and their valuable comments on the report. This work has been partially supported by Polish National Science Center under the Grant agreement 2019/33/B/ST6/02011. AG has been also supported by Polish National Science Center under the Grant agreements 2020/37/N/ST6/02220. We would like to thank the organizers of *BMW Group Quantum Computing Challenge* for providing us with the exemplary dataset used in this manuscript.

Data availability The data supporting the findings of this study are available at https://doi.org/10.5281/zenodo.6012261.

Declarations

Conflict of interest The authors have no competing interests to declare.

References

1. Bayerstadler A, Becquin G, Binder J, Botter T, Ehm H, Ehmer T, Erdmann M, Gaus N, Harbach P, Hess M, et al. Industry quantum computing applications. EPJ Quantum Technol. 2021;8(1):25.
2. Tilly J, Chen H, Cao S, Picozzi D, Setia K, Li Y, Grant E, Wossnig L, Rungger I, Booth GH, Tennyson J. The variational quantum eigensolver: a review of methods and best practices. Physics Reports. 2022;986:1–128.
3. Farhi E, Goldstone J, Gutmann S. A quantum approximate optimization algorithm. arXiv preprint arXiv:1411.4028 (2014).
4. Das A, Chakrabarti BK. Colloquium: quantum annealing and analog quantum computation. Rev Mod Phys. 2008;80(3):1061.
5. Johnson MW, Amin MH, Gildert S, Lanting T, Hamze F, Dickson N, Harris R, Berkley AJ, Johansson J, Bunyk P,

et al. Quantum annealing with manufactured spins. Nature. 2011;473(7346):194–8.
6. Featured Applications: D-Wave. https://www.dwavesys.com/learn/featured-applications. Accessed 10 Feb 2022.
7. Domino K, Kundu A, Salehi Ö, Krawiec K. Quadratic and higher-order unconstrained binary optimization of railway rescheduling for quantum computing. Quantum Inf Process. 2022;21(9):1–33.
8. Salehi Ö, Glos A, Miszczak JA. Unconstrained binary models of the travelling salesman problem variants for quantum optimization. Quantum Inf Process. 2022;21(2):1–30.
9. Domino, Krzysztof, Mátyás Koniorczyk, Krzysztof Krawiec, Konrad Jałowiecki, Sebastian Deffner, and Bartłomiej Gardas. "Quantum annealing in the NISQ era: railway conflict management." Entropy 25, no. 2: 191 (2023).
10. Yarkoni S, Alekseyenko A, Streif M, Von Dollen D, Neukart F, Bäck T. Multi-car paint shop optimization with quantum annealing. In: 2021 IEEE international conference on quantum computing and engineering (QCE). IEEE; 2021. p. 35–41.
11. Mugel S, Kuchkovsky C, Sanchez E, Fernandez-Lorenzo S, Luis-Hita J, Lizaso E, Orus R. Dynamic portfolio optimization with real datasets using quantum processors and quantum-inspired tensor networks. Phys Rev Res. 2022;4(1):013006.
12. Kurowski K, Weglarz J, Subocz M, Różycki R, Waligóra G. Hybrid quantum annealing heuristic method for solving job shop scheduling problem. In: International conference on computational science. Springer; 2020. p. 502–15.
13. Genin SN, Ryabinkin IG, Izmaylov AF. Quantum chemistry on quantum annealers. arXiv preprint arXiv:1901.04715 (2019).
14. Teplukhin A, Kendrick BK, Tretiak S, Dub PA. Electronic structure with direct diagonalization on a D-wave quantum annealer. Sci Rep. 2020;10(1):1–11.
15. Mato K, Mengoni R, Ottaviani D, Palermo G. Quantum molecular unfolding. Quantum science and technology 7(3), p.035020 (2022).
16. Aspronі L, Caputo D, Silva B, Fazzi G, Magagnini M. Accuracy and minor embedding in subqubo decomposition with fully connected large problems: a case study about the number partitioning problem. Quantum Mach Intell. 2020;2(1):1–7.
17. Jiang S, Britt KA, McCaskey AJ, Humble TS, Kais S. Quantum annealing for prime factorization. Sci Rep. 2018;8(1):1–9.
18. Arya A, Botelho L, Cañete F, Kapadia D, Salehi Ö. Applications of quantum annealing to music theory. Cham: Springer International Publishing; 2022. p. 373–406.
19. Luckow A, Klepsch J, Pichlmeier J. Quantum computing: towards industry reference problems. Digitale Welt. 2021;5(2):38–45.
20. Tiepelt MK, Singh TR. Finding pre-production vehicle configurations using a Max-SAT framework. In: 18th international configuration workshop; 2016. p. 117.
21. Walter R, Kübart T, Küchlin W. Optimal coverage in automotive configuration. In: International conference on mathematical aspects of computer and information sciences. Springer; 2015. p. 611–26.
22. BMW Group. Optimizing Production of Test Vehicles. https://crowd-innovation.bmwgroup.com/apps/IMT/UploadedFiles/00/f_b20f223487b934a44f2d92db76044434/210818_UC1_Config.pdf?v=1643293520. Accessed 10 Feb 2022.
23. Hybrid Solver for Constrained Quadratic Models [WhitePaper]. https://www.dwavesys.com/media/rldh2ghw/14-1055a-a_hybrid_solver_for_constrained_quadratic_models.pdf. Accessed 10 Feb 2022.
24. Gu J, Purdom PW, Franco J, Wah BW. Algorithms for the satisfiability (SAT) problem: a survey. Technical report, Cincinnati University of Department of Electrical and Computer Engineering (1996).
25. Cook SA. The complexity of theorem-proving procedures. In: Proceedings of the third annual ACM symposium on theory of computing; 1971. p. 151–8.
26. Garey, M. R.; Johnson, D. S. Computers and Intractability: A Guide to the Theory of NP-Completeness. A Series of Books in the Mathematical Sciences. San Francisco, Calif.: W. H. Freeman and Co. ISBN 0-7167-1045-5. MR 0519066 (1979).
27. Kirkpatrick S, Gelatt CD, Vecchi MP. Optimization by simulated annealing. Science. 1983;220(4598):671–80.
28. Kelley JE. The cutting-plane method for solving convex programs. J Soc Ind Appl Math. 1960;8:703–12.
29. El Lawler DEW. Branch-and-bound methods: a survey. Oper Res. 1966;14:699–719.
30. Mitchell S, OSullivan M, Dunning I. PuLP: a linear programming toolkit for python. Auckland: The University of Auckland; 2011. p. 65.
31. Forrest J, Lougee-Heimer R. CBC user guide. In: Emerging theory, methods, and applications. INFORMS; 2005. p. 257–77.
32. Gurobi Optimization, LLC. Gurobi optimizer reference manual. https://www.gurobi.com (2021). Accessed 10 Feb 2022.
33. Lucas A. Ising formulations of many NP problems. Front Phys vol.2, p.5 (2014).
34. Choi V. Minor-embedding in adiabatic quantum computation: I. The parameter setting problem. Quantum Inf Process. 2008;7(5):193–209.
35. Shi Y, Reich D, Epelman M, Klampfl E, Cohn A. An analytical approach to prototype vehicle test scheduling. Omega. 2017;67:168–76.
36. Kundu A. iitis/bmw_vehicle_opt: v1.0.0. 10.5281/zenodo.6012261 (remove this reference as we put this on the data availability statement)
37. Glos A, Krawiec A, Zimborás Z. Space-efficient binary optimization for variational quantum computing. npj Quantum Inf. 2022;8(1):1–8.
38. Tabi Z, El-Safty KH, Kallus Z, Hága P, Kozsik T, Glos A, Zimborás Z. Quantum optimization for the graph coloring problem with space-efficient embedding. In: 2020 IEEE international conference on quantum computing and engineering (QCE). IEEE; 2020. p. 56–62.
39. Campbell C, Dahl E. QAOA of the highest order. In: 2022 IEEE 19th international conference on software architecture companion (ICSA-C). IEEE; 2022. p. 141–6.
40. Mohammadbagherpoor H, Dreher P, Ibrahim M, Oh YH, Hall J, Stone RE, Stojkovic M. Exploring airline gate-scheduling optimization using quantum computers. arXiv preprint arXiv:2111.09472 (2021).
41. Bakó B, Glos A, Salehi Ö, Zimborás Z. Near-optimal circuit design for variational quantum optimization. arXiv:2209.03386 (2022).
42. Singh TR, Rangaraj N. Generation of predictive configurations for production planning. In: Configuration workshop; 2013. p. 79–86.
43. Walter R, Küchlin W. ReMax—a MaxSAT aided product (re-)configurator. In: Configuration workshop; 2014. p. 59–66.