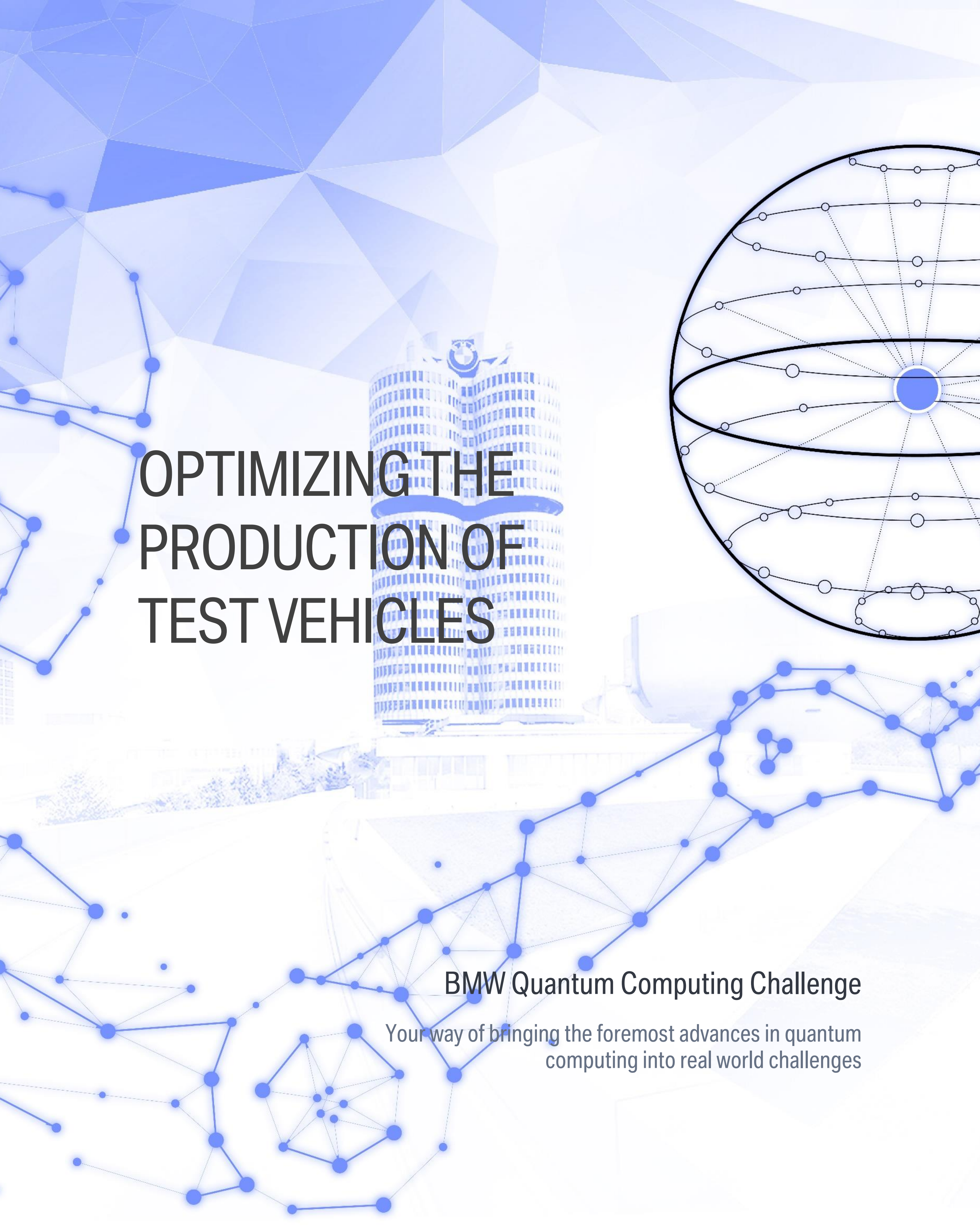
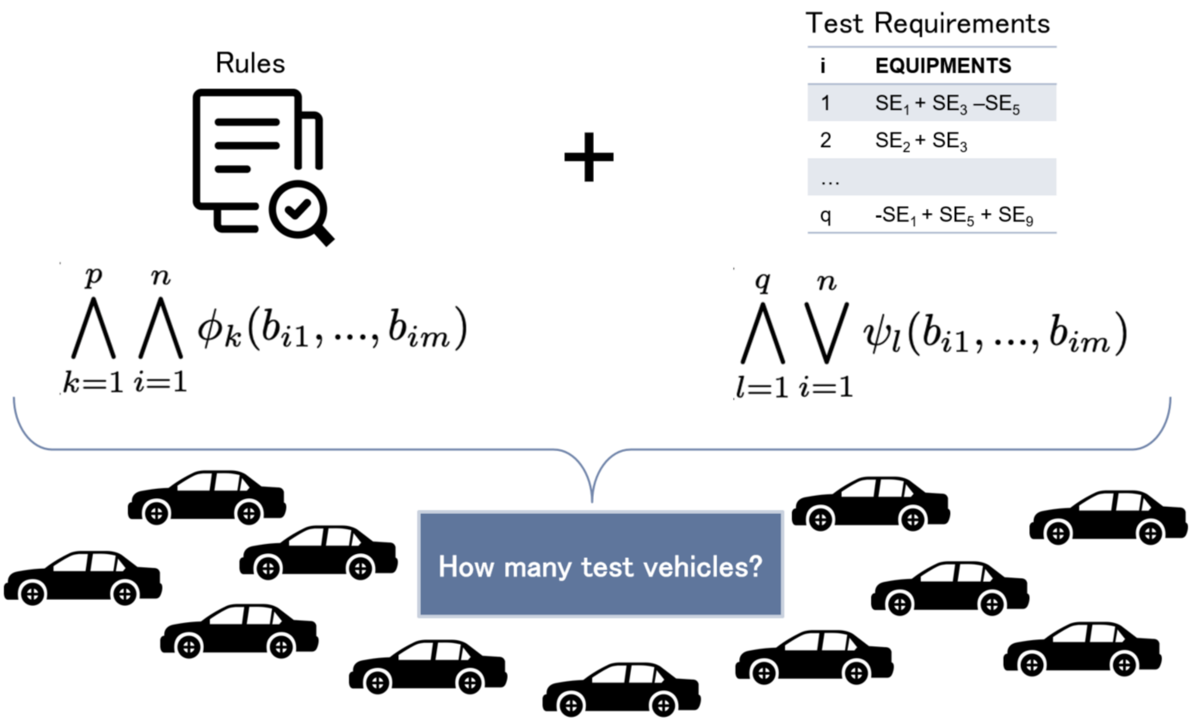


OPTIMIZING THE PRODUCTION OF TEST VEHICLES

BMW Quantum Computing Challenge

Your way of bringing the foremost advances in quantum computing into real world challenges



Test Requirements

i	EQUIPMENTS
1	$SE_1 + SE_3 - SE_5$
2	$SE_2 + SE_3$
…	
q	$-SE_1 + SE_5 + SE_9$

$$\bigwedge_{k=1}^{p} \bigwedge_{i=1}^{n} \phi_k(b_{i1}, \ldots, b_{im})$$

$$\bigwedge_{l=1}^{q} \bigvee_{i=1}^{n} \psi_l(b_{i1}, \ldots, b_{im})$$

Motivation & Business Value

Before new vehicle models can be released for official production, various tests must be performed on pre-series vehicles. These tests include evaluating the possibility of producing specific components and validating the model's functionality. The desired set of tests should be distributed and performed to minimize the number of test vehicles produced by the manufacturer, and as a result, to reduce the cost of production. Thus, even for an optimal combination of features, the problem requires balancing costs and test coverage. Ideally, the decision-maker could estimate the number of possible tests for a certain number of vehicles (which directly translates to costs). One main complication for the real-world process is that different departments require separate tests, which cannot be performed simultaneously. This implies an additional scheduling problem to determine which tests are performed when.

Problem formulation

Let us consider n potential test vehicles $v_1, \dots, v_n$, with the configuration of each vehicle v_i being unambiguously defined by the presence or absence of any of the m available features $a_1, \dots, a_m$.

For each vehicle v_i let the boolean / binary variable b_{ij} denote the presence or absence of feature a_j. We let 1 represent "true" and 0 represent "false" throughout this paper, thus $b_{ij} = 1$ if and only if vehicle v_i contains feature a_j.

Buildability constraints

For several logical and technical reasons, not all combinations of features are feasible. A logical reason example: a car cannot contain both the 4 and the 6 cylinder engine. A technical reason could be that a specific engine does not allow a manual gearshift.

Let us assume that we have p such "configuration constraints". Since they apply to m features and are identical for all n test cars they can be written in the form

$$\phi_k(b_{i1}, \dots, b_{im}) = 1\,, \qquad \text{for } i = 1, \dots, n \quad \text{and} \quad k = 1, \dots, p \tag{1}$$

where the ϕ_k are boolean expressions in the m variables $b_{i1}, \dots, b_{im}$.

Example: If a configuration rule stipulates that each vehicle must either have features 1 and 2 or it must have feature 3, this translates into the Boolean expressions

$$\bigwedge_{i=1}^{n} \phi_{example}(b_{i1}, \dots, b_{im}) = \bigwedge_{i=1}^{n} (b_{i1} \wedge b_{i2}) \vee b_{i3}\,. \tag{2}$$

One can think of the total number of constraints (1) as governing the "buildability" of the n test vehicles. An equivalent formulation of (1) is

$$\bigwedge_{i=1}^{n} \bigwedge_{k=1}^{p} \phi_k(b_{i1}, \dots, b_{im})\ . \tag{3}$$

Test constraints

In the testing phase a number of tests need to be performed, and each test requires a vehicle with a certain constellation of features. Let us consider q tests and let their requirements on the features of car i be given by

$$\psi_l(b_{i1}, \dots, b_{im}) = 1 \text{ for } l = 1, \dots, q, \tag{4}$$

where each ψ_l is a boolean expression in the m variables $b_{i1}, \dots, b_{im}$. Vehicle i thus fulfills the testing requirement l if and only if

$$\psi_l(b_{i1}, \dots, b_{im}) = 1. \tag{5}$$

For example, if one requests (at least) one test car having feature k_1 but no feature k_2 enabled, the boolean expression would be:

$$\bigvee_{i=1}^{n} \psi_{example}(b_{i1}, \dots, b_{im}) = \bigvee_{i=1}^{n} (b_{ik_1} \wedge \neg\, b_{ik_2}) = 1, \tag{6}$$

Note that all cars need to fulfil the buildability constraints, which leads to the overall AND-clause (see Eq. 2); whereas it's enough to have at least one car that fulfils the testing requirements (OR-clause, see Eq. 6).

The above setting can give rise to various mathematical problems that fall into the SAT (satisfiability problems) or MAX SAT class. The detailed description for both classes is given below.

SAT problems

The basic SAT problem can be formulated as follows: "Is there any configuration of features for a number of n cars so that they satisfy the buildability constraints (1) and that all testing requirements (4) are met by at least one car". Mathematically, it can be represented by following equation:

$$\left(\bigwedge_{k=1}^{p} \bigwedge_{i=1}^{n} \phi_k(b_{i1}, \dots, b_{im}) \right) \wedge \left(\bigwedge_{l=1}^{q} \bigvee_{i=1}^{n} \psi_l(b_{i1}, \dots, b_{im}) \right) = 1\,. \tag{7}$$

We refer to (7) as the basic problem.

There may be test requirements that need to be met by more than one vehicle in a real-world case, say by d vehicles. This situation can be accommodated by copying the testing requirement in question so that it appears d times in the requirement set (4). Obviously, it will not solve the problem since direct copying of a requirement is inconsequential. However, it is now possible to add simple features ensuring that those (identical) requirements are not fulfilled by the same car. In fact, adding $\lceil \log_2 d \rceil$ new features and, thus, $n \cdot \lceil \log_2 d \rceil$ new variables makes those d requirements mutually exclusive. This approach leaves the structure of the optimization problem unchanged.

If one requests only the smallest n where equation (7) holds, a bisection approach can be adopted in this case. To that extend, one may start with q vehicles since building one car for each test requirement will solve the SAT problem (in the condition it is solvable). It leaves the minimum number of vehicles n in the range between zero and q. Adapting the bisection method for finding roots of a function [1] one continues with checking if $\frac{q}{2}$ vehicles fulfil the requirement. If this is the case, we reduce the range to $\frac{q}{2}$ to 0. Otherwise, the solution has to be in the range from q to $\frac{q}{2}$. Repeating this process, one eventually arrives at the optimal value for n by at most $\lceil \log_2 q \rceil$ applications of the SAT problem.

MAX-SAT problem

As stated in the motivation, it is essential to know the number of vehicles required to perform a given number of tests. In reality, however, the business needs to judge how many tests it wants to perform before the production starts, as its cost limits the number of possible tests. Thus, a more relevant approach is to compute the number of tests that can be performed for a given number of vehicles.

It leads to a Weighted Partial MAX-SAT problem: "Find the constellation of features for n test vehicles, each complying with the configuration rules, so weighted number q of the fulfilled testing requests is maximized."

In this case, the optimization problem can be written as :

$$\underset{b_{ij}}{\text{maximize}}\, f(b_{11}, \dots, b_{nm}) = \sum_{l=1}^{q} w_l \cdot \bigvee_{i=1}^{n} \psi_l(b_{i1}, \dots, b_{im}) \tag{8}$$

subject to the configuration constraints (1), with $w_l > 0$ being the weight of the l-th test for $l = 1, \dots, q$. The weights may vary, making specific tests have a higher priority than others.

A solution must fulfill the set of configuration constraints as described in (1). These hard (or partial) constraints can be accounted for in the objective function f by multiplying them with large weights.

Scheduling problem

All specified tests need to be performed after the vehicles are produced, and the test cars are specified. Reducing the number of test cars could still make the scheduling problem harder and even impossible to solve in the presence of additional scheduling constraints. In this case, more cars might be necessary to schedule all tests. Among the possible scheduling constraints one may consider:

- Each test has a time frame in which it needs to be performed:

$$t_l^{start}, t_l^{end} \text{ for } l = 1, \dots, q\,;$$

- Some tests destroy (parts of) the car and therefore make it impossible to perform (some) other tests afterwards;
- There is a limited amount of test engineers, each test needs one or multiple engineers available (some tests may request different specialists as well).

Solving both problems at once (when is the test performed and which car is built & used for it) increases the complexity of the problem, but might be necessary if the scheduling constraints are very limiting and they impact the number of produced cars.

Scheduling problems can be formulated as SAT problems, but the formulation is usually very cumbersome and expressive. Typically, the Integer Linear Programming or Constraint Programming formulations are preferred. In practice, SAT or Integer based formulations can be used when a discrete-time representation is employed for the scheduling problem. Otherwise, heuristics or genetic algorithms become a common choice.

Test problem and classical benchmarks

The problems stated above belong to the so-called Satisfiability (SAT) and Maximum Satisfiability (MAX-SAT) classes.

Problem (8) deviates from the classical MAX-SAT setting and results in a Weighted Partial Max-SAT problem. This arises as the boolean expressions describing the configuration restrictions defined in (1) are "hard" (partial), i.e. must be met. Whereas for the expressions defining the test cases in (4), the maximum possible amount of restrictions should be fulfilled optimally based on their assigned weights. The maximization happens only on these boolean expressions, which describe the objective function. In any case, the problem complexity is the same as one of MAX-SAT, which is NP-hard, and even APX-complete.

Despite the unfavorable complexity of both the SAT and the MAX-SAT problems, it is known that branch and bound algorithms (as CDCL [2]) in combination with heuristics can, in practice, provide good results for some (MAX)-SAT problems (to be established for the problem case).

Test problem

The problem example is inspired by the BMW Series 2 Gran Coupe. The provided description is based on the actual numbers and constraints formulated for this model. It, thus, represents the real complexity arising in a productive setting. Note, however, that the complexity of vehicles and production systems is ever increasing. Future instances of this problem will grow in their complexity as well. The data describing the optimization problem consists of two parts, representing the test requirements and buildability constraints.

Test Requirements

The first part contains the requirements necessary to conduct a test. In the test example, a list of $q = 672$ tests requirements is specified. This list contains the necessary number of vehicles, which a specific test needs to be performed with, and the vehicle features which must or must not be configured. An extract of this list is given in listing 1:

```
3 :     F160 ~F26 ~F37
3 :     ~F26 ~F37 ~F160
3 :     ( F26 | F37 ) F160
3 :     ( F26 | F37 ) ~F160
1 :     ~F13 ~F14 ~F124
1 :     F124 ~F13 ~F14
1 :     ~F13 ~F14 ~F124
1 :     F124 ~F13 ~F14
```

LISTING 1 EXAMPLE SPECIFICATIONS OF THE TEST REQUIREMENTS. THE INTEGER AT THE BEGINNING OF THE LINE DESCRIBES HOW OFTEN THE TEST IS REQUIRED. THE TEXT AFTER THE COLON REPRESENTS THE DEFINITIONS OF NECESSARY FEATURES OR EXCLUDED FEATURES.

Buildability Constraints

The second part includes the buildability constraints. For the test example, a rulebook with approximately 5k rules is used as the optimization basis. Note that the rules are dynamic in a real-world scenario, which means they are only valid for a certain time frame. To simplify the approach, the rules can be selected as being valid at the same specific date. In the data for the test problem, the following buildability constraints are modeled:

1. Each car has exactly one type;

2. Allowed features per type: each car type has a list of allowed features for its type:

```
T0 :   F4 F21 F288 F124 F160 F199 F450 F321 F399 F131 F146 F465 F38
       F425 F211 F349 F53 F294 F435 F281 F304 F320 F157 F432 F153
       F398 F314 ...
T1 :   F21 F465 F38 F425 F211 F349 F294 F368 F153 F398 F314 F275 F344
       F389 F210 F299 F269 F372 F378 F151 F262 F243 F399 F131 F435
       F281 F304 F320 ...
```

LISTING 2 EXTRACT FROM THE BUILDABILITY CONSTRAINTS SHOWING THE ALLOWED FEATURES FOR TWO VEHICLE TYPES.

3. Feature groups: only one of the features in this group can be true at the same time; for example, only one steering wheel can be configured for a vehicle:

```
group features (only one of them can be active at the same time, each line a
group):
F393 F125
F350 F330 F322 F326 F331 F321 F319 F317 F349 F318 F343 F320 F323
F344 F325 F356 F357 F332 F327 F353 F346 F347 F345 F354 F355 F351
F352 F328 F329
F83 F84
F445 F443 F444 F446
```

LISTING 3 EXTRACT FROM THE BUILDABILITY CONSTRAINTS SHOWING FOUR FEATURE GROUPS WHOSE FEATURES ARE MUTUALLY EXCLUSIVE

4. Configuration rules: these are implication rules that are specific for each type. Given a vehicle type and the active features set, a list of other features is forced or forbidden.

```
T23 :   F23 => F240
T23 :   F240 => F3 | F22 | F23 | F78 | F82
T22 :   F312 & ~F19 & ~F373 & ~F380 => F232 | F233
T22 :   F238 => F101 | F102 | F103 | F104 | F106 | F107 | F108 | F109 | F110
        | F117 | F118 | F119 | F120 | F421
```

LISTING 4 EXTRACT FROM THE BUILDABILITY CONSTRAINTS SHOWING TWO CONFIGURATION TOOLS FOR TWO TYPES RESPECTIVELY.

Scheduling Constraints

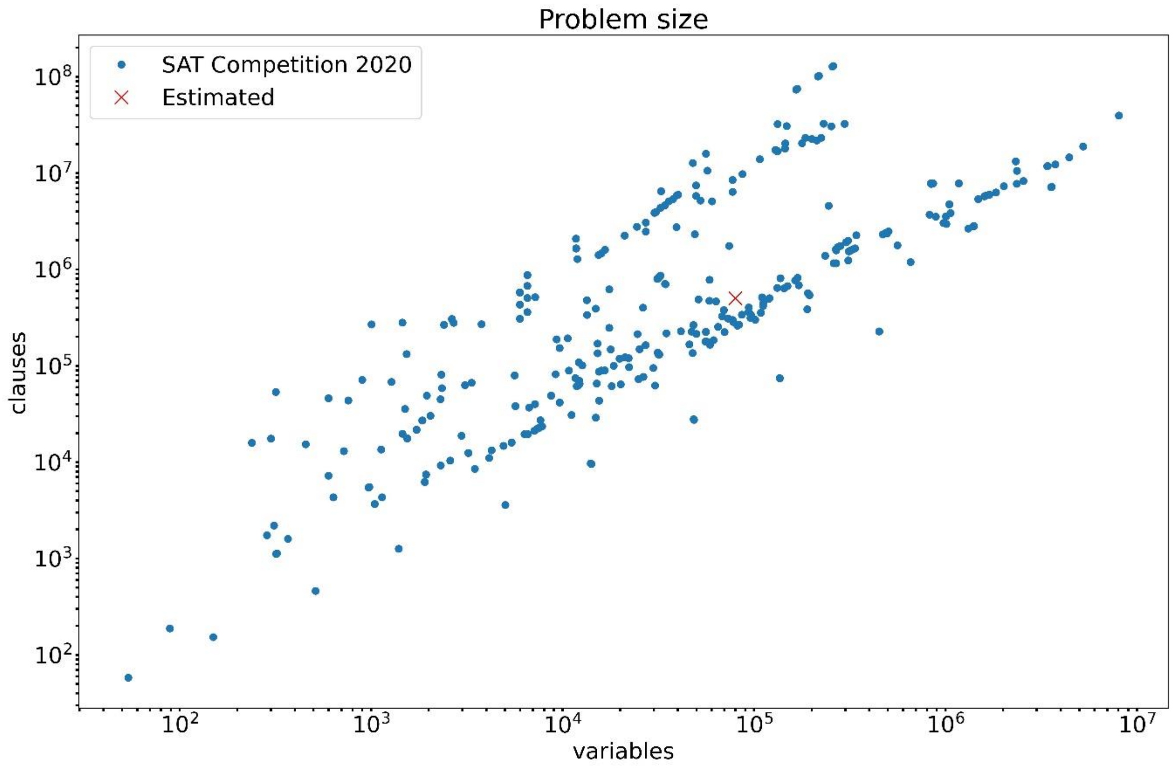
To include the constraints described in *Scheduling problem* no specific model is enforced. A simple way of modelling the complexities is as follows:

- scheduling is done on day-to-day basis;

- tests are assumed to take exactly one day;
- every day each car can only be used for one test;
- the number of tests performed per day is restricted by a constant integer number (corresponding to the number of test engineers);
- a random group (integer number) is assigned to each test; tests with a higher test group are not allowed to be performed after a test with a lower test group (e.g. a sound test could be in group 3, a test destroying one part of the vehicle in group 2 and a full crash test in group 1).

Problem Size

Based on the outline above, the problem results in a formulation with approximately 800 variables and 5k clauses per car. When considering 100 test cars, it yields 80k variables and 500k clauses. The images below show the problem size (red cross) in comparison to the problems of the SAT Competition 2020, which had a time limit of 90 min per problem



For the SAT problem, the instance size seems to be in a solvable regime. MAX-SAT instances are solved in practice by repeated calling an SAT-Oracle, which usually is an implementation of a fast SAT solver (for example RC2 the best MAX-SAT solver 2018/19 uses Glucose as its base-solver [3]). The run time of a MAX-SAT solver for an equally sized problem should, therefore, be expected much higher than for a SAT problem.

Results of classical implementation

An implementation based on pysat was conducted. Real data for tests and buildability constraints was used, resulting in a problem with 500 Boolean variables and 1.2k clauses per car.

SAT- / MAX-SAT-Problem

To solve the test problem with classical solvers, the buildability constraints are modelled in the following manner:

- each car has exactly one type:
  - the type is added as another Boolean variable;
  - one clause is added to ensure that at least one type is used;
  - for each pair of types, another clause is added to ensure that one of them is set to false, thus, only one type can be set at once;
- allowed features per type:
  - modeled by a clause for each feature: if the feature is true, the type that allows this feature is true as well;
- feature groups:
  - modeled similar to the type restrictions, via pairs of features in the same group;
- configuration rules:
  - the most challenging aspect, as not all can be directly written in the CNF (conjugated normal form) format;
  - more complex rules were converted to CNF by introducing additional variables via the Tseitin algorithm.

For 100 cars, the problem can be solved in a few seconds. A linear search counting down from 100 revealed the solution that at least 60 cars are needed to perform all the specified 750 tests.

On the other hand, the MAX-SAT problem was not solvable in a reasonable time with the chosen approach. A different solver or heuristics are assumed still to provide a fast solution to the MAX-SAT problem.

Scheduling Problem

To estimate the scalability of the combined problem, a small test implementation based on constraint programming was developed. The implementation for the classical SAT problem (*Results of classical implementation*) was converted to constraint programming[1]. The run-time was slightly higher than the previous SAT-based implementation but still solvable in a few seconds.

This basic constraint programming implementation was then extended with the scheduling part. For this implementation, no real data of the scheduling constrains were available. Therefore, random data was used instead. Run-time experiments with a variable number of tests were performed. For all experiments, the scheduling horizon was set to 100 days and up to 10 tests could be performed per day. Test groups were assigned randomly on a scale of 1 to 5.

As expected, the observed run-time scales strongly with the number of tests:

- for 350 tests and up to 60 cars, the current formulation results in 150k variables, which is solvable in 80 minutes;

[1] The Constraint solver CP-SAT from Google OR-Tools was used for the implementation

- for 550 tests and up to 60 cars, the current formulation results in 270k variables, which is solvable in 10 hours;
- on the test laptop, the full problem with 700 tests wasn't solvable in less than 24 hours.

Overall, the combined problem seems to be at the limit of current constraint solvers. An improved implementation using more computational power will probably provide a solution to some cases in a reasonable time, especially if combined with heuristics.

Applicability of QC Algorithms

Several different approaches are viable and discussed in the literature. In this section a short summary of potential research directions is given.

The plain SAT problem can be solved using Grover's Algorithm [4], one of the earliest and the most famous quantum algorithms. It has a run time of $O(\sqrt{N})$, needs as many qubits as variables and a circuit scales linearly with the number of clauses. The MAX-SAT problem can be solved classically by calling a SAT oracle multiple times. The expected return includes the number of conflicting clauses in the case of an unsatisfiable problem. Grover's algorithm does not provide conflicting clauses and can not be easily used as a subroutine in a MAX-SAT solver.

QAOA extends the idea of Grover's algorithm to combinatorial optimization problems. Grover's algorithm can be seen as a special case of the QAOA algorithm [5]. QAOA can therefore be used to solve MAX-SAT instances. However, the performance is strongly dependent on the problem and thus exhibits a similar critical clause density as classical algorithms [5]. Nevertheless, QAOA is still an active research topic and several ideas exist that might improve the performance (e.g. removing the classical outer part for a variational free version [6], changing the optimization techniques for the outer part [7] or finding better matching mixing Hamiltonians [8]).

Solving the plain SAT problem directly using Grover's search should yield a faster algorithm (higher success rate). However, a variation of QAOA might allow implementation with a smaller number of gates that could potentially be implemented on near-term devices [9].

Quantum walks could represent another approach for the problem. They have shown to bring a square root speed up in detecting and finding a solution to a problem compared to a classic random walk. Classical random walk algorithms can be applied to finding a solution to a boolean formula and thus solve a SAT problem. Further, a method based on quantum walks to gain speedup for backtracking algorithms (which are the basis for many state-of-the-art SAT solvers) has been presented by Montanaro [10]. And a possible implementation of this approach was introduced in [11].

Using Quantum Annealing to solve the MAX-SAT problem might be an alternative. For example, [12] introduces schemes for embedding SAT problems onto the Ising model, which can then be solved on quantum annealing hardware.

Research Proposal

The test vehicle optimization problem is a scientifically challenging problem with high commercial value for BMW. In its basic formulation, it can be expressed as a SAT-problem, which is well solvable with classical approaches. Extensions to the problem (MAX-SAT and the combination with the scheduling problem) increase the complexity heavily, resulting in models that can exhibit a prohibitively long run-time (*Results of classical implementation*). Still, in practice, certain approximations or simplifications can provide reasonable solutions.

Research Focus

Quantum computing algorithms are guaranteed to scale better for the classical SAT problem, but no such guarantees could be found for the extensions. Nevertheless, quantum algorithms might provide an advantage for more complex models, specifically in combination with the scheduling problem. Only experiments with real hardware will provide clear evidence if this is the case. The proposed research contribution, thus, constitutes an approach to solve the MAX-SAT problem with a quantum computer or hybrid strategies involving both classical and quantum resources. The special interest here lies in establishing a theoretically defined advantage that will be provided by future hardware. Approaches with especially low time-to-market requirements or approaches, including the scheduling problem, will be particularly appreciated. The main metric considered for a new approach evaluation is the innovative character of the solution and how efficient it scales (i.e. a solution with lower hardware requirements).

Test Data

For testing and validation purposes, a dataset is provided. The dataset must be considered as a reference to a real-world problem. It contains the information described in *Test problem*:

- buildability constraints;
- test requirements.

Due to the current hardware limitations, a simplified model (i.e. a subset or artificially generated data) can be employed.

Acknowledgement

This document was created with the help of Deloitte Consulting and the feedback from AWS.

Appendix A: List of Symbols / Abbreviations

Symbol	Meaning
n	Number of test vehicles
m	Number of different available features
v_i	Vehicle i
a_i	Feature i
b_{ij}	Vehicle v_i has feature a_j - boolean / binary variable
$\phi_k(b_{i1}, \dots, b_{im})$	configuration constrain k for vehicle i as a Boolean expression of features
p	number of configuration constraints
$\psi_l(b_{i1}, \dots, b_{im})$	test constrain l for vehicle i as a Boolean expression of features
q	number of test constraints

Appendix B: Detailed Example

We assume the following features, a car might have:

Car feature	Meaning
a_1	small engine
a_2	big engine
a_3	strong sub-woofer
a_4	seat heating
a_5	comfort seating

These features have the following constrains: A car needs to have a small or a big engine, but not both.

$$\bigwedge_{i=1}^{n} \phi_{engine}(b_{i1}, \dots, b_{im}) = \bigwedge_{i=1}^{n} (b_{i1} \vee b_{i2}) \wedge (\neg b_{i1} \vee \neg b_{i2}) \tag{9}$$

A car can, but doesn't have to, have seat heating. That can be the basic seat heating or through the comfort seating (which includes a different heating option), but not both.

$$\bigwedge_{i=1}^{n} \phi_{seating}(b_{i1}, \dots, b_{im}) = \bigwedge_{i=1}^{n} \neg b_{i4} \vee \neg b_{i5} \tag{10}$$

In addition, there are a few tests that should be performed:

$$\psi_{breaks}(b_{i1}, \dots, b_{im}) = b_{i1} \tag{11}$$

$$\psi_{breaks2}(b_{i1}, \dots, b_{im}) = b_{i2} \tag{12}$$

$$\psi_{Noise\_level}(b_{i1}, \dots, b_{im}) = b_{i2} \wedge b_{i3} \tag{13}$$

$$\psi_{Bass\&Seat\_interaction}(b_{i1}, \dots, b_{im}) = b_{i3} \wedge b_{i4} \tag{14}$$

$$\psi_{Power\_consumption}(b_{i1}, \dots, b_{im}) = b_{i2} \wedge b_{i3} \wedge b_{i5} \tag{15}$$

To perform all tests, you need 3 tests cars. With two test cars, all but one test can be performed.

BMW GROUP

References

[1] Wikipedia, “Bisection method,” https://en.wikipedia.org/wiki/Bisection_method, [Online; accessed June-2021]..

[2] J. P. M. Silva and K. A. Sakallah, “ GRASP-a new search algorithm for satisfiability,” ICCAD, vol. 96, pp. 20-227, 1996.

[3] A. Ignatiev, A. Morgado and J. Marques-Silva, “RC2: an effi-cient MaxSAT solver,” Journal on Satisfiability, Boolean Modeling and Computation, vol. 11.1, p. 53–64, "=!).

[4] L. K. Grover, “ A fast quantum mechanical algorithm for database search,” Proceedings of the twenty-eighth annual ACM symposium on Theory ofcomputing, p. 212–219, 1996.

[5] V Akshay et al., “Reachability deficits in quantum approximate optimiza-tion,” Physical review letters, vol. 124.9, p. 090504, 2020.

[6] M. Streif and M. Leib, “Training the quantum approximate opti-mization algorithm without access to a quantum processing unit,” Quantum Science and Technology, vol. 5.3, p. 034008, 2020.

[7] Zhihui Wang et al., “X y mixers: Analytical and numerical results for thequantum alternating operator ansatz,” Physical Review A, vol. 101.1, p. 012320, 2020.

[8] A. Bärtschi and S. Eidenbenz, “Grover mixers for QAOA: Shift-ing complexity from mixer design to state preparation,” IEEE International Conference on Quantum Computing and Engineering (QCE), p. 72–82, 2020.

[9] Z. Jiang, E. G. Rieffel and Z. Wang., “Near-optimal quantumcircuit for Grover’s unstructured search using a transverse field,” Physical Review A, vol. 95.6, p. 062317, 2017.

[10] A. Montanaro, “Quantum walk speedup of backtracking algorithms,” 2016.

[11] S. Martiel and M. Remaud, “Practical Implementation of a Quantum Backtracking Algorithm,” Springer International Publishing, 2020, p. 597–606.

[12] Z. Bian, F. Chudak, W. Macready, A. Roy, R. Sebastiani and S. Varotti, “Solving SAT (and MaxSAT) with a quantum annealer: Foundations, encodings, and preliminary results,” Information and Computation, vol. 275, p. 104609, 2020.