Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

A Use-Case-Oriented Benchmark Framework for Classical, Quantum, Hybrid, and Quantum-Inspired Optimization

Dominik Leib

dominik.leib@itwm.fraunhofer.de

Fraunhofer Institute for Industrial Mathematics

Tobias Seidel

Fraunhofer Institute for Industrial Mathematics

Sebastian Velten

Fraunhofer Institute for Industrial Mathematics

Michael Bortz

Fraunhofer Institute for Industrial Mathematics

Research Article

Keywords: Quantum Optimization, Benchmarking, Quantum-Hybrid, Classical Optimization

Posted Date: June 15th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-9845326/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: No competing interests reported.

A Use-Case-Oriented Benchmark Framework for Classical, Quantum, Hybrid, and Quantum-Inspired Optimization

Dominik Leib[1*], Tobias Seidel[1], Sebastian Velten[1], Michael Bortz[1]

[1*]Department Optimization, Fraunhofer Institute for Industrial Mathematics, Fraunhofer-Platz 1, Kaiserslautern, 67663, Rhineland-Palatinate, Germany.

*Corresponding author(s). E-mail(s): dominik.leib@itwm.fraunhofer.de;
Contributing authors: tobias.seidel@itwm.fraunhofer.de;
sebastian.velten@itwm.fraunhofer.de;
michael.bortz@itwm.fraunhofer.de;

Abstract

As interest in quantum and related optimization paradigms continues to grow, fairly and systematically comparing their performance across practical problem settings remains challenging. This challenge arises because these approaches often target different problem classes, use different mathematical representations, and are affected differently by modeling choices. As a result, observed performance depends not only on the solver itself, but also on how a problem is formulated, reformulated, and translated into a solver-specific input language.

This work presents a publicly available and extensible benchmarking framework that unifies advances in quantum, quantum-hybrid, quantum-inspired, and classical optimization by enabling consistent comparison across use cases, solvers, and models. We address the diversity of solver "languages"—from mixed-integer programming (MIP) formulations for branch-and-bound methods over quadratic unconstrained binary optimization (QUBO) models for quantum annealers to constrained quadratic models (CQM) for hybrid approaches — as well as the influence of modeling choices, such as binary encodings and penalty formulations, on both performance and solution fidelity. By supporting systematic variation across the three dimensions of use case, solver, and model, the framework reveals trade-offs that single-dimension benchmarks overlook, clarifies when native formulations are preferable, and enables rigorous evaluation of practical combinatorial optimization problems. The two case studies, vehicle routing and maximum cut, illustrate this principle: their contrasting outcomes show that

the effectiveness of a solver depends strongly on how well the chosen model representation matches the solver backend and the use case.

Keywords: Quantum Optimization, Benchmarking, Quantum-Hybrid, Classical Optimization

1 Motivation

Benchmarking has long been a central methodology for evaluating and comparing the performance and solution quality of computational methods across technologies and over time [1, 2]. By providing standardized tasks, metrics, and experimental protocols, benchmarks enable fair, reproducible, and interpretable comparisons between algorithms, implementations, and hardware platforms (see e. g. [3]).

As emerging paradigms such as quantum, quantum-hybrid, and quantum-inspired optimization mature alongside increasingly powerful classical solvers, this benchmarking role becomes even more critical. In particular, practical quantum devices have introduced a new class of computational platforms that promise performance advantages for selected use cases. Advances across quantum, quantum-hybrid, and quantum-inspired optimization have produced a rapidly growing ecosystem of solvers and execution backends, which has triggered an active line of benchmarking efforts for quantum and related technologies [4–6].

Existing benchmark suites for quantum and quantum-inspired optimization largely focus on characterizing hardware or restricted algorithm families. Hardware- and stack-centric frameworks such as Q-score [7], AppQSim [8], and SupermarQ [9] define a small set of canonical workloads and report device-level performance metrics across platforms, thereby providing valuable, standardized stress tests for gate-based and annealing architectures. Their studies start from concrete tasks, but typically fix a single algorithm family and model type, so that modeling choices remain largely implicit.

Closer to our work, QUARK [10] and QOBLIB [11] provide application-driven benchmark environments for combinatorial optimization problems. QUARK formulates two industrial decision problems as reusable "pen-and-paper" benchmarks and implements a modular stack to compare classical and quantum(-inspired) solvers on fixed QUBO-style formulations, while QOBLIB offers a curated library of challenging problem classes with MIP and QUBO encodings to systematically track progress toward quantum advantage. Both frameworks represent important steps toward practice-oriented benchmarking; however, they primarily restrict attention to very specific combinatorial optimization problems and expose only a small, fixed set of formulations per use case, so that the interaction between alternative models (e. g., QUBO vs. MIP vs. CQM) and heterogeneous solver technologies is only partially captured. These limitations motivate the perspective adopted in this work.

The central question is therefore not only whether a method can solve an instance, but how different technologies compare on the same real-world optimization problem in terms of solution quality, time-to-solution, robustness across runs, and modeling

overhead. A key obstacle to meaningful comparison is that optimization technology is rarely applied to abstract problem classes in isolation; it is applied to *use cases*. Use cases come with domain-specific parameters, constraints, and objective structure, and these details strongly influence which formulation is appropriate. In practice, benchmarking follows a pipeline: generate or collect representative instances for a use case, choose a mathematical *model* for each solver technology, and then execute solver runs while recording relevant metrics. Most benchmark suites implicitly collapse parts of this pipeline—e. g., by fixing a single formulation or by focusing on one solver family—and thereby mix algorithmic performance with modeling choices. We therefore view benchmarking as a three-dimensional design space:

- **Use cases**, which define domain semantics and instance distributions,
- **Models**, which are concrete formulations of a use case in a specific modeling language,
- **Solvers**, which operate on specific model types and expose different trade-offs.

A benchmark that fixes one dimension therefore risks attributing outcomes to solvers that are in fact driven by model choices, or vice versa. We address this by introducing a *use-case-oriented benchmark framework* that separates instance creation, model construction, and solver execution.

Our main contribution is to make this three-dimensional perspective explicit and actionable. For this, we present a use-case-oriented benchmark framework that supports systematic comparison across multiple model representations and solver technologies on identical problem instances. This allows us to analyze not only which solver performs best for a given model and use case, but also how strongly performance depends on the chosen formulation and its alignment with the solver backend. For each generated problem instance, the framework stores the underlying use-case parameters together with one or more derived models in a library. Benchmark runs then operate on these libraries and evaluate a configurable list of solver–model pairs. This setup makes it possible to study not only which solver performs best, but also which modeling choices make a solver competitive on a given use case and where translation overhead or model-induced scaling become limiting factors.

This view reflects how optimization is actually deployed. Branch-and-bound solvers are primarily designed for MIP-style formulations, while quantum annealers and many quantum-inspired heuristics naturally operate on QUBO representations. Even within a single use case, valid formulations can therefore differ substantially in variable count, constraint handling, and numerical behavior. The numerical case studies (cf. Sections 3 and 4) show that the alignment between model structure and solver design can have a decisive impact on performance. In the vehicle-routing study, Gurobi performs best on the native linear mixed integer program (MIP) formulation, whereas the Leap Hybrid constrained quadratic model (CQM) solver benefits from the binary constrained variant; in the Maximum-Cut study, by contrast, Gurobi is more competitive on the quadratic unconstrained binary optimization problem (QUBO) formulation under the chosen runtime budget. This highlights that solver capabilities alone do not determine

performance; rather, outcomes depend on the interaction between solver architecture, model representation, and benchmark regime. The software implementation accompanying this framework is publicly available on GitHub [12].

The remainder of this work is organized as follows. Section 2 describes the key elements of the benchmark framework, including use cases, solvers, the benchmark procedure, and artifact persistence. Sections 3 and 4 then demonstrate the framework on two case studies. Section 3 studies the vehicle routing use case and compares D-Wave's Leap Hybrid solvers with classical baselines. Section 4 demonstrates the framework on the Maximum-Cut problem and compares D-Wave's quantum annealer with classical approaches. Detailed model descriptions are contained in Section A and the supplementary material.

2 Framework Description

Our benchmark framework operationalizes the three-dimensional perspective introduced above in a modular workflow from instance generation to result interpretation. In practice, it supports two closely related use modes. First, users can assemble benchmark studies directly from the existing implementation by selecting supported use cases, model formulations, and solver backends and then executing standardized comparisons. Second, the code base can be extended in a modular way: new use cases can be added, additional model formulations can be implemented for existing or new use cases, and further solver backends can be connected through the common interfaces. This allows practitioners to start from a concrete optimization use case and build further standardized comparisons on top of the existing framework. Detailed examples, usage instructions, and implementation guidelines for extending the framework with additional use cases, models, or solvers are provided in the documentation and the accompanying code base. In the following, we give a conceptual description of the framework and its core benchmarking workflow.

As shown in Figure 1, a benchmark study in the framework starts with the generation of problem instances for a given use case or of entire libraries of such instances. These instances are then transformed into one or more model representations and stored together with the corresponding instantiation of the use-case parameters. Based on these stored artifacts, solver–model combinations can be benchmarked systematically, and the resulting solutions can be evaluated with respect to use-case-specific performance measures. This workflow allows a clear separation between use-case definition, instance generation, model construction, solver execution, and result analysis. The remainder of this section describes these stages and the main features provided by the framework.

2.1 Use Cases and Instance Generation

At the highest abstraction level, the framework is organized around optimization use cases. In the codebase, these use cases are implemented as classes whose names end in `Problem`. Each use case defines the semantic structure of a class of optimization instances through a parameterized description. A concrete problem instance is uniquely specified by its *use-case-parameters*. Based on this abstraction, the framework

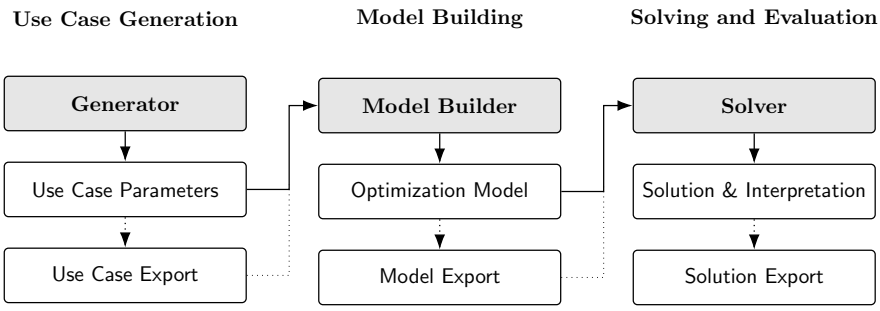


Fig. 1: Workflow and exported artifacts from generating to solving a use case instance with the framework, together with exported artifacts.

provides generators for individual problem instances as well as for larger collections of instances that can be used in systematic benchmarking studies. This separation between use case definition and instance generation is important for two reasons. First, it allows experiments to be conducted on families of structurally comparable instances rather than on isolated hand-crafted examples. Second, it provides a uniform entry point for the subsequent modeling pipeline, since all later stages operate on the same use-case parameters regardless of the target model representation. As a result, the framework supports reproducible benchmark design across multiple use cases and instance sizes. Correspondingly, instance generators produce individual problem instances, library generators produce libraries of such instances, and model builders create the corresponding models for the targeted model type.

The current version of the framework covers five discrete optimization use cases that span canonical graph benchmarks and decision problems motivated by industrial settings. On the graph side, we include Maximum Cut (Max-Cut) and Maximum Independent Set (MIS), two classical NP-hard problems that are widely used as benchmarks in combinatorial optimization and complexity theory [13–15]. For routing, we consider a stop-capacitated vehicle routing variant as a standard representative of vehicle routing models [16, 17]. In addition, the framework contains two application-oriented use cases: an inbound container scheduling use case inspired by container terminal operations [18, 19] and a discrete portfolio optimization use case with budget, logical coupling, and pairwise discount effects [20, 21]. Table 1 lists all use cases together with their code-level names and short semantic descriptions; detailed model formulations and use case parameters for the case-study use cases are given in Appendix A, while the remaining use cases are documented in the supplementary material.

2.2 Model Types and Models

Given the use case parameters of a concrete problem instance, the framework features the formulation of mathematical models that expose different structural properties to the solvers. In the current implementation we support three major model classes:

Use Case	Explanation
MaximumCutProblem	The Maximum Cut Problem is, given a graph $G = (V, E)$ with edge weights, to partition V into two sets V_1 and V_2 such that the sum of weights of edges with endpoints in different sets is maximized.
MaximumIndependentSetProblem	Given a graph (V, E) with vertex weights, select an independent subset, that is $V' \subseteq V$ such that no two vertices in V' are adjacent, and the total weight of V' is maximized.
VehicleRoutingProblem	The stop-capacitated vehicle routing problem generalizes the Traveling Salesperson Problem to multiple vehicles with a bounded number of customer stops per route, where each node besides a destined depot represents a customer to be served. Given a depot and a fleet of vehicles, the objective is to construct routes that serve all customers exactly once while respecting the maximum number of stops per vehicle and minimizing the total route length.
ContainerSchedulingProblem	The goal of container scheduling (inbound scheduling) is to assign each container a time t and a cargo bay b for unloading so that no two containers occupy the same bay at the same time (i. e., at most one container per bay and time slot). Containers have different unloading durations, there is a global time limit, and grouped containers may need to be assigned consistently to bay groups. The implemented objective minimizes the sum of unloading start times, subject to bay-feasibility restrictions and the scheduling constraints.
PortfolioOptimizationProblem	Given a set of brands with several admissible facings, each carrying a cost and a profit, choose at most one facing per brand to maximize discounted total profit subject to a budget. Additional logical coupling constraints may apply, and selected brand pairs can induce pairwise profit discounts.

Table 1: Implemented use cases in the benchmark framework with short descriptions.

mixed-integer programming (MIP) formulations, quadratic unconstrained binary optimization (QUBO) models, and constrained quadratic models (CQM). All three represent the same underlying combinatorial decision problem, but differ in variable types, how constraints are handled, and which solver backends they are compatible with. We sketch the key modeling features for each approach in the following.

A MIP model represents the use case as a finite-dimensional optimization problem with linear objective and linear constraints, where a subset of variables is restricted to be integral. Classical mixed-integer programming formulations with linear objectives and constraints are a standard workhorse in combinatorial optimization; see, e. g., [22] or [23]. A generic MIP formulation has the form

$$\min_{x \in \mathbb{R}^n} \; c^\top x \quad \text{s.t.} \quad Ax \leq b,\; A^{\text{eq}} x = b^{\text{eq}},\; x_i \in \mathbb{Z}\; \forall i \in I, \tag{1}$$

where $c \in \mathbb{R}^n$ is the objective coefficient vector, A, A^{eq} and b, b^{eq} encode inequality and equality constraints, and $I \subseteq \{1, \ldots, n\}$ indexes the integer variables. In our benchmark framework, MIP formulations are the canonical input for classical branch-and-bound type solvers such as Gurobi, which provide global optimality guarantees up to a prescribed optimality gap. Constraints remain explicit in (1) and do not need to be embedded into the objective via penalty terms, which typically leads to well-scaled models with clear feasibility information.

A QUBO represents the optimization problem solely in terms of binary decision variables with a quadratic objective and no explicit constraints. QUBO models provide a unifying framework for binary quadratic problems and are widely used as a target representation for quantum and quantum-inspired solvers; see [24] and [25] for surveys and modeling guidelines. A standard QUBO reads

$$\min_{x \in \{0,1\}^n} \; x^\top Q x + c^\top x + \gamma, \tag{2}$$

where $Q \in \mathbb{R}^{n \times n}$ is a (typically symmetric) coefficient matrix, $c \in \mathbb{R}^n$ is a linear term, and $\gamma \in \mathbb{R}$ is a constant offset. "Unconstrained" means that all logical or combinatorial constraints of the original use case are encoded as penalty terms in the entries of Q and c, respectively. QUBO models are the native input for many quantum and quantum-inspired backends, including quantum annealers, simulated annealing, and QAOA-based solvers. Within the framework, a QUBO builder maps the use case parameters of a problem instance to (2) using use-case-specific binary encodings and penalty scales, which allows us to study how such modeling choices influence solver performance and solution quality.

Finally, CQMs occupy a middle ground between classical MIP formulations and purely unconstrained QUBOs. CQMs, as implemented in D-Wave's Ocean toolchain, generalize QUBO models by allowing explicit constraints and mixed variable types while retaining quadratic structure; see the Ocean documentation on CQM [26] and the description of the Leap Hybrid CQM solver [27]. A generic CQM can be written as

$$\begin{aligned} \min_{x} \quad & x^\top Q_0 x + c_0^\top x + \gamma \\ \text{s.t.} \quad & x^\top Q_k x + c_k^\top x \le b_k, \quad k \in \mathcal{K}, \\ & x_j \in \{0,1\} \quad \forall j \in \mathcal{B}, \\ & x_j \in \mathbb{Z} \quad\;\; \forall j \in \mathcal{I}, \\ & x_j \in \mathbb{R} \quad\;\; \forall j \in \mathcal{C}, \end{aligned} \tag{3}$$

where $\mathcal{K}$ is a finite index set of constraints, γ a constant offset and $\mathcal{B}, \mathcal{I}, \mathcal{C}$ partition the variable indices into binary, integer, and continuous subsets. In our framework, CQM formulations are used as input to hybrid quantum-classical solvers that natively accept such models (e. g., D-Wave's Leap Hybrid CQM solver). Compared to QUBOs, CQMs avoid encoding all constraints as penalties and thus reduce sensitivity to penalty calibration, while still exposing a quadratic structure that is compatible with

annealing-style and related heuristic methods. This makes CQMs particularly useful for use cases where constraint handling is nontrivial but a purely linear MIP formulation would obscure the natural quadratic interactions of the problem.

To connect the abstract model types to executable solver backends, each formulation is created in the modeling language associated with its representation and then stored in a unified `Model` wrapper. Concretely, QUBO formulations expose a binary quadratic model object, MIP formulations expose a mixed-integer model object, and CQM formulations expose a constrained quadratic model object. These backend-specific objects are not handled directly throughout the benchmark logic. Instead, they are encapsulated together with shared metadata such as the model name, instance identifiers, and interpretation routines in the corresponding model class. Solver wrappers can therefore interact with models through a common interface and access the underlying backend object only when needed inside their `solve` method. This design decouples benchmark orchestration from solver-specific APIs while still allowing each solver to operate on the native representation required by its backend.

For each model classes MIP, QUBO, and CQM, the framework implements concrete use-case-specific models. Table 2 summarizes the currently supported model variants together with compact abbreviations that we use when referring to models independently of the solver backend.

2.3 Solvers

To enable fair and reproducible comparisons across heterogeneous technologies, the framework wraps a diverse set of solver backends behind a thin, uniform interface. Each backend operates on persisted model artifacts of type MIP, QUBO, or CQM, exposes a common API for setting solver parameters, and emits a common solution record containing objective value, runtime, solution status, solver settings, and a backend-specific information field. A compatibility layer ensures that only valid solver–model pairs are executed, while additional diagnostics such as hardware timing or raw solver logs can be preserved when a backend exposes them. The integrated portfolio spans three families: classical exact and near-exact methods for MIP, quantum and quantum-inspired samplers for QUBO, and hybrid solvers that combine classical search with quantum sampling and accept constrained quadratic models. Table 3 provides a compact overview of the currently supported backend wrappers; detailed notes per solver follow thereafter.

As for the classical baselines, Gurobi (GRB) is a state-of-the-art mixed-integer programming solver based on branch-and-bound with cutting planes, presolve, and primal/dual heuristics. It provides global optimality guarantees up to a user-specified optimality gap and is particularly effective on well-scaled linear formulations with explicit constraints [22, 23]. With Gurobi QUBO (GRBQ) we built another wrapper that solves QUBOs as binary (generally nonconvex) quadratic models using global branch-and-bound and convexification/cutting techniques by Gurobi. It serves as an exact baseline for small to medium, not overly dense QUBOs, but can degrade on dense or poorly scaled penalty encodings; careful penalty calibration is often crucial [24, 25]. Simulated annealing (SA) is a classical heuristic based on the Metropolis acceptance rule on the QUBO energy landscape. SA offers a fast, robust baseline

Use Case	Type	Abbr.	Explanation
Maximum Cut	MIP	MCMIP	Native linear mixed-integer formulation of the Maximum-Cut problem.
	QUBO	MCQUBO	Binary quadratic model of Maximum-Cut.
	CQM	MCCQM	Constrained quadratic formulation of Maximum-Cut with explicit auxiliary variables and constraints.
Maximum Independent Set	MIP	MISMIP	Native linear mixed-integer formulation of the Maximum Independent Set problem.
	QUBO	MISQUBO	Binary quadratic model of Maximum Independent Set.
	CQM	MISCQM	Constrained quadratic formulation of Maximum Independent Set.
Vehicle Routing	MIP	VRPMIP	Native arc-based linear MIP formulation of the stop-capacitated vehicle routing problem.
	MIP	VRPMIPB	Binary position-indexed MIP formulation corresponding more closely to the QUBO encoding with penalties encoded as constraints.
	QUBO	VRPQUBO	Binary quadratic model of the vehicle routing problem.
	CQM	VRPCQM	Native arc-based constrained quadratic formulation with explicit routing constraints.
	CQM	VRPCQMB	Binary/quadratic constrained formulation corresponding to the position-indexed routing encoding.
Container Scheduling	MIP	CSMIP	Mixed-integer formulation of the inbound container scheduling problem.
	QUBO	CSQUBO	Binary quadratic model of container scheduling using penalty terms.
	CQM	CSCQM	Constrained quadratic model of container scheduling with explicit feasibility constraints.
Portfolio Optimization	MIP	POMIP	Mixed-integer formulation of the portfolio optimization problem with explicit budget and logic constraints.
	QUBO	POQUBO	Binary quadratic model of the portfolio optimization problem.
	CQM	POCQM	Constrained quadratic formulation of the portfolio optimization problem.

Table 2: Overview of the concrete models currently supported in the benchmark framework.

Wrapper	Abbr.	Type	Explanation
GurobiSolver	GRB	MIP	Gurobi is a commercial state-of-the-art MIP solver.
GurobiQUBOSolver	GRBQ	QUBO	Solves QUBOs to optimality using Gurobi.
SimulatedAnnealingSolver	SA	QUBO	Solves QUBOs via simulated annealing, a probabilistic metaheuristic for binary optimization.
DWaveQuantumSolver	DW	QUBO	D-Wave's quantum annealer; solves QUBOs via quantum annealing.
LeapHybridBQMSolver	LBQM	QUBO	A commercial, proprietary hybrid solver on the Leap platform; leverages D-Wave quantum resources within a hybrid workflow.
LeapHybridCQMSolver	LCQM	CQM	A commercial, proprietary hybrid solver on the Leap platform; leverages D-Wave quantum resources within a hybrid workflow and accepts the possibility to encode constraints.
QAOASolver	QAOA	QUBO	Solves QUBOs by the Quantum Approximate Optimization Algorithm, an iterative variational strategy based on a p-layer quantum circuit and classical parameter optimization; the current wrapper is simulator-based but can in principle be extended to quantum backends.

Table 3: Overview of solver wrappers available in the framework with short explanations.

with controllable runtime (via reads/sweeps) but no optimality guarantees; solution quality depends on the cooling schedule and initialization, so multiple seeded runs are typically aggregated; see [28, 29] for detailed descriptions of the algorithm.

Beyond the classical baselines, we include three families of non-classical solvers. First, we use D-Wave quantum annealer (DW) via the Ocean SDK, which natively solves QUBO models by quantum annealing [30, 31]. Our DW wrapper takes the QUBO representation produced by the model builders, converts it to a binary quadratic model, performs minor-embedding onto the hardware graph, and then samples from the corresponding annealing schedule; the solver record preserves the timing information returned by the backend so that hardware usage can be analyzed separately. Second, we evaluate D-Wave's cloud-based Leap Hybrid solvers, which combine large-scale classical metaheuristics with calls to D-Wave quantum processing units (QPUs). We employ both Leap binary quadratic model solver (LBQM) for unconstrained QUBOs and Leap constrained quadratic model solver (LCQM) for constrained quadratic models. This allows us to contrast penalty-based QUBO encodings with native constraint handling in CQM while using the same backend technology; various descriptions and technical reports of the Leap framework are to be found in [26, 27, 32, 33]. Third, we integrate a variational gate-based Quantum Approximate

Optimization Algorithm (QAOA) solver, which constructs a p-layer QAOA circuit from the QUBO Hamiltonian and uses a classical optimizer to tune the variational parameters; in the current implementation this wrapper is instantiated with a simulator backend by default, but the interface can in principle be extended to quantum backends. An introduction of the QAOA procedure can be found in [34]. All three solver families are accessed through the unified benchmark `Solver` interface, and for each run we persist common quantities such as solution quality and wall-clock time together with any backend-specific timing or diagnostic information that is available.

2.4 Benchmark Procedure, Metrics, and Code Extensibility

Besides supporting reproducible benchmark studies, the framework is designed as extensible research software. In particular, a user can add a new use case by implementing the corresponding problem and generation components, add further model formulations for an existing or new use case by connecting them to the appropriate model interface, or integrate an additional solver backend by implementing the common solver interface for one of the supported model types. This code-level modularity is central to the framework, since it allows a practitioner who enters with a concrete optimization use case to connect new formulations and solver backends without changing the overall workflow.

We now describe the benchmark routine used by the framework to combine persisted problem instances, model artifacts, and solver backends into a reproducible evaluation workflow. The central executable object is a *solver–model pair*, that is, an ordered pair $(\mathcal{S}, \mathcal{M})$ consisting of a concrete solver wrapper $\mathcal{S}$ and a concrete stored model representation $\mathcal{M}$ for the same underlying problem instance, such that $\mathcal{S}$ is a valid interface for the model type of $\mathcal{M}$. The feasible solver–model pairs can be derived by the corresponding Type columns in Tables 2 and 3. Hence, the benchmark interface accepts a library together with a finite set of such solver–model pairs and executes only combinations that are valid under the predefined compatibility map between solver types and model types.

As illustrated in Figure 2, the benchmark routine operates on a persisted library state rather than on transient intermediate objects. For each problem instance stored in the library, it iterates over compatible solver–model pairs that are not yet marked as completed in the checklist, loads the corresponding stored model representation, executes the solver, and immediately persists the resulting solution file while updating the checklist. This makes benchmark runs interruptible and resumable at the level of individual solver–model executions and allows metrics and visualizations to be generated afterwards from the persisted solution files. Concretely, the routine proceeds as follows:

1. For each stored problem instance, the benchmark loads the corresponding use-case parameters and the requested model.
2. For each admissible solver–model pair, the benchmark loads the matching stored model representation and executes the solver wrapper on that model.

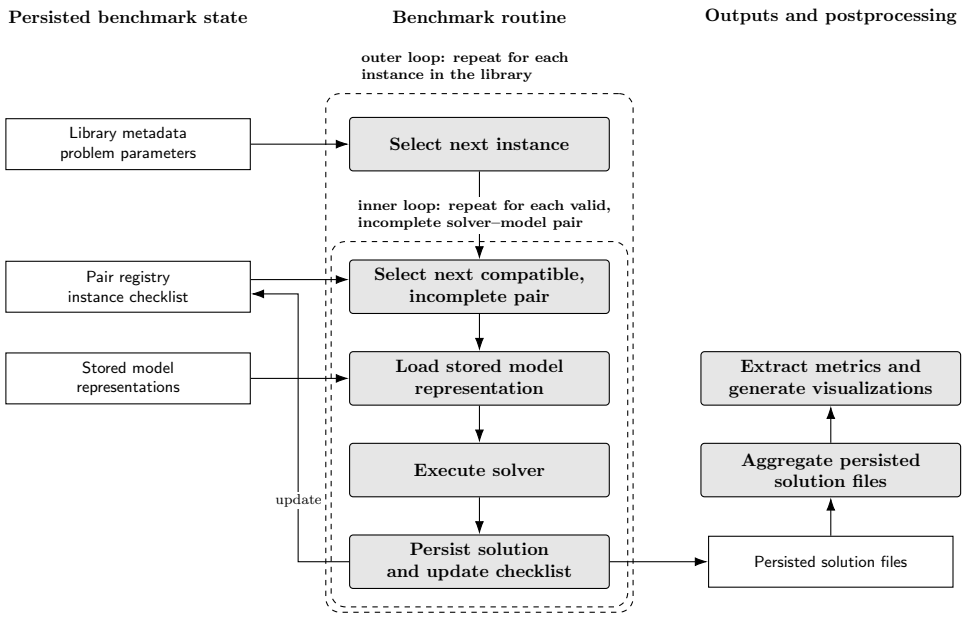


Fig. 2: Sketch of the benchmark routine. For each instance in the persisted library and for each compatible, incomplete solver–model pair, the routine loads the stored model representation, executes the solver, persists the resulting solution file, and updates the checklist. The persisted solution files can subsequently be aggregated for metric extraction and visualization.

3. The number of runs is pair-dependent. Deterministic Gurobi-based MIP and Gurobi-based QUBO runs are executed once, whereas stochastic and hybrid solver calls for the same pair and instance may be repeated.
4. After every individual run, the resulting solution artifact is written to disk immediately and the run status is updated in the checklist. This makes the procedure interruptible and resumable for each run.

This workflow yields a persistent benchmark state that can be extended incrementally. If a study is interrupted, rerun with additional solver–model pairs, or extended by more stochastic repetitions, the benchmark continues from the existing checklist instead of recomputing completed runs. The benchmark writes four classes of artifacts:

1. *Library metadata*, which enumerate the stored problem instances, the available model names, and for each instance the filenames of the corresponding parameter and model artifacts.
2. A *solver–model pair registry*, which stores the benchmark configuration, i.e., the selected pair titles together with the underlying solver class, the chosen parameter settings, and the associated model name.

3. A per-instance *checklist*, that records which solver–model pairs have already been executed for a given instance and which concrete solution filenames belong to these completed runs.
4. One *solution file* per run, optionally supplemented by raw backend log files such as Gurobi logs for post-hoc analysis of bound trajectories and time-to-target behavior.

Each solution file follows a common schema. It stores the solution status, a human-readable explanation, a use-case-specific interpretation of the solution, the raw variable assignment returned by the solver, the objective value, the measured wall-clock runtime, the problem and model instance identifiers, the solver settings used for that run, and an additional information field for backend-specific diagnostics. This common record structure is solver-independent and is generated uniformly by the abstract solver interface.

Consequently, the benchmark tracks two layers of quantities. First, it records *generic run metrics* that are available for every solver–model pair, most importantly runtime, solver-side objective value, use-case-level problem value, feasibility status, and run metadata. Second, it records *backend-specific information* exposed by individual wrappers, such as additional timing or hardware-usage information for D-Wave-based runs. These quantities are not left for users to implement from scratch in standard studies: the current library already provides import and visualization utilities for runtimes, problem values, feasibility information, and backend-specific timing quantities such as QPU access time. In addition, Gurobi log files can be parsed to derive bound information and time-to-target style metrics for comparison plots. The provided plotting utilities therefore already support standard result visualizations such as runtime plots, problem-value plots, repeated-run ranges, and optional bound overlays. Problem-specific quality measures can still be derived from the persisted solution interpretations during downstream analysis, and the metric and plotting layer is itself extensible because additional stored fields or log-derived quantities can be exposed through the same import and visualization interfaces. This separation keeps execution, storage, and later evaluation conceptually distinct while preserving enough information for detailed post-processing. Because benchmark state is persisted in libraries, pair registries, checklists, and per-run solution files, newly implemented models and solver wrappers can be incorporated into existing studies without invalidating previously computed artifacts.

3 Case Study: Vehicle Routing

To demonstrate how the benchmark framework can be used to compare different models of the same optimization problem across solver classes, we consider a vehicle routing case study. The goal is not only to compare classical and hybrid solvers, but also to study how strongly solver performance depends on the chosen model. We compare five solver–model combinations. As classical baselines, we use Gurobi on two mixed-integer models of the vehicle routing problem: a native linear arc-based MIP model (VRPMIP) and a binary position-indexed MIP model (VRPMIPB), corresponding to the QUBO

model. On the hybrid side, we use LBQM on the QUBO model (VRPQUBO), as well as LCQM on both a binary CQM model (VRPCQMB) and a native arc-based CQM model (VRPCQM). All formulations follow the Miller-Tucker-Zemlin (MTZ) subtour elimination strategy [35] and are based on well-known formulations in the literature, see e. g. [16]. Detailed formulations of the vehicle-routing models used in this case study are given in Appendix A.

3.1 Setup

We generated a compact instance library of vehicle routing instances, parameterized by the triplet (N, M, K), where N denotes the number of nodes in the graph representing the customer network, M the number of vehicles, and K the maximum number of customer stops per vehicle. We consider instance sizes $N \in \{30, 40, 50, 60, 70, 80\}$ with corresponding numbers of vehicles $(N, M) \in \{(30, 2), (40, 2), (50, 3), (60, 3), (70, 4), (80, 4)\}$. The node locations are sampled i.i.d. uniformly in the unit square $[0, 1]^2$, where the Euclidean distances define the routing cost matrix entries. The stop capacities are chosen according to $K = \left\lceil \frac{N-1}{M} \cdot 1.1 \right\rceil$, which yields the configurations

$$(N, M, K) \in \{(30, 2, 16), (40, 2, 22), (50, 3, 18), (60, 3, 22), (70, 4, 19), (80, 4, 22)\}.$$

For each configuration, we generate two random instances, resulting in a total of 12 benchmark instances. All solver–model combinations were executed with the same nominal time limit of 25 seconds per run. Deterministic Gurobi-based MIP runs were executed once per instance, while stochastic solver–model combinations (LBQM on QUBO, and LCQM on both CQM variants) have three runs per instance. The resulting benchmark therefore remains small enough to be reproducible under limited compute resources while still covering multiple problem sizes and several structurally different models. Table 4 lists all parameters for the case study.

3.2 Results

Across all tested sizes, all solver–model combinations returned feasible solutions. However, the achieved objective values differ substantially, and the gap becomes more pronounced as the instance size increases, as Figure 3 shows.

The strongest overall results were obtained by Gurobi on the native arc-based MIP model (VRPMIP). In contrast, Gurobi on the binary position-indexed MIP model (VRPMIPB) performed substantially worse, especially for larger instances. For example, at $N = 60$, the native Gurobi MIP model achieved a mean objective value of 7.42, while the binary Gurobi MIP model reached 27.78. This indicates that the classical MIP solver is much better matched to the native linear model than to the binary position-indexed encoding.

Among the hybrid approaches, the binary/quadratic CQM model consistently produced the best objective values. At $N = 80$, the mean objective values are: 10.31 for VRPMIP (Gurobi), 13.51 for LCQM on VRPCQMB, 28.23 for LBQM on VRPQUBO, 38.73 for LCQM on VRPCQM, and 47.57 for VRPMIPB (Gurobi). Thus, while the

Parameter	Meaning	Value
problem	Use case class	VehicleRoutingProblem
N	Instance sizes	$\{30, 40, 50, 60, 70, 80\}$
M	Number of vehicles per size	$\{30 : 2, 40 : 2, 50 : 3, 60 : 3, 70 : 4, 80 : 4\}$
capacity_rule	Maximum number of customer stops per vehicle	$\lceil \frac{N-1}{M} \cdot 1.1 \rceil$
K	Resulting stop capacities	$\{30 : 16, 40 : 22, 50 : 18, 60 : 22, 70 : 19, 80 : 22\}$
node_sampling	Sampling of node coordinates	i.i.d. uniform on $[0, 1]^2$
distance_metric	Edge cost between nodes	Euclidean distance
solver_model_pairs	Combinations of models and solvers benchmarked	(GRB, VRPMIP), (GRB, VRPMIPB), (LBQM, VRPQUBO), (LCQM, VRPCQMB), (LCQM, VRPCQM)
num_problems_per_size	Instances generated per size	2
num_instances	Total instances in library	12
random_seed	Python RNG seed	42
num_reruns_qubo	Reruns for stochastic QUBO-based solver calls	3
GRB time_limit_s	Time limit GRB in seconds	25
LBQM time_limit_s	Time limit LBQM in seconds	25
LCQM time_limit_s	Time limit LCQM in seconds	25
metric	Optimization metric	sum of route lengths

Table 4: Configuration of the Vehicle Routing case study.

classical solver benefits from the linear model, the LCQM backend benefits from the binary constrained model.

These results indicate that solver performance is tightly coupled to the model structure. In our experiments, the Gurobi backend performs best on the native linear MIP model, whereas LCQM performs best on the binary constrained model.

4 Case Study: Maximum Cut

Unlike the preceding vehicle routing case study, where GRB performed best on the native linear model, the Maximum-Cut study shows the opposite pattern under the chosen benchmark budget: the quadratic model is particularly favorable for GRBQ. This complements the vehicle routing example and illustrates that no model family is uniformly superior; performance depends on the solver–model pairing. In the following, we compare four solver–model combinations on a common family of weighted graph instances: GRB on a Max-Cut MIP model, GRBQ on a Max-Cut QUBO model, SA on the QUBO model, and DW on the same QUBO model. The linear MIP formulation follows a standardized node-based approach, as widely applied in the literature [15, 22, 36]. The QUBO formulation follows the standard binary encoding of Max-Cut, as discussed for example in [24, 25, 37]. The concrete Maximum-Cut formulations underlying these comparisons are summarized in Appendix A.

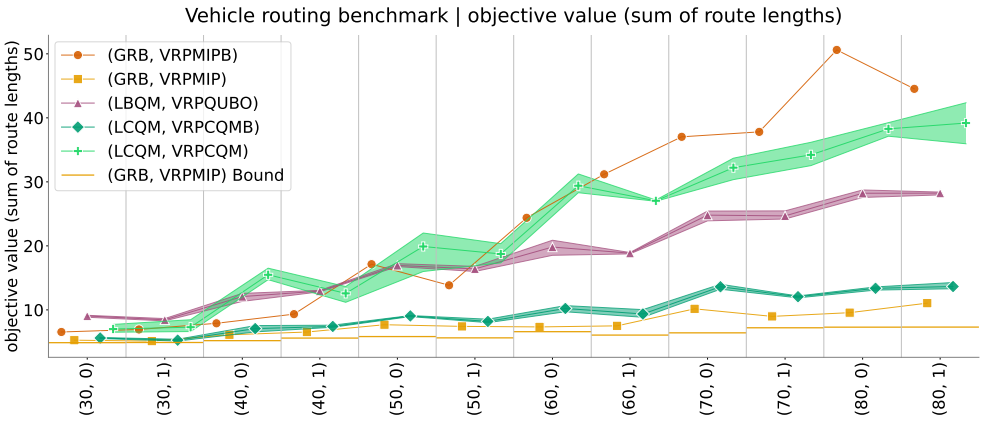


Fig. 3: Objective values for the vehicle routing case study across all instances and solver-model combinations, where lower values correspond to shorter total route lengths. The instances are separated by gray vertical bars and each instance is noted as (N, i), where N is the number of nodes and i the index of the instance of this size. For repeated stochastic or hybrid runs, markers denote mean values and shaded bands indicate the observed min–max range. Best observed bounds for (GRB, VRPMIP) are shown as horizontal lines in the corresponding color.

4.1 Setup

We generated a compact instance library of complete weighted graphs with instance sizes $N \in \{50, 60, 70, \ldots, 160\}$. For each size, two random instances were generated, resulting in a total of 24 instances. Edge weights were sampled i.i.d. from a lognormal distribution, clipped at quantile 0.995, and normalized to the interval $(0, 1]$, using parameter $\sigma = 2.0$. The native MIP and QUBO formulations were benchmarked with Gurobi using a time limit of 20 seconds per run. Gurobi runs were executed once per instance, whereas SA and DW were repeated three times per instance. For SA, we used 100 reads and 50 sweeps. For DW, we used 100 reads on sampler Advantage_system6.4.

In addition to cut values, we record runtime and, for D-Wave Quantum, also the reported QPU access time. Table 5 summarizes the case-study configuration.

4.2 Results

Across all tested sizes, the achieved cut values and the associated timing behavior differ increasingly as the instance size grows. Figure 4 shows the objective values.

For the smaller tested instances, the two QUBO-based classical baselines already achieve very similar solution quality, while the native MIP model remains close but begins to lag under the 20-second budget. D-Wave Quantum already shows a visible quality gap in this regime. For example, at $N = 50$, the mean cut values are 37.85 for $(\mathrm{GRBQ}, \mathrm{MCQUBO})$, 37.83 for $(\mathrm{SA}, \mathrm{MCQUBO})$, 37.73 for $(\mathrm{GRB}, \mathrm{MCMIP})$, and 36.50 for $(\mathrm{DW}, \mathrm{MCQUBO})$.

Parameter	Meaning	Value
problem	Use case class	MaximumCutProblem
N	Instance sizes	$\{50, 60, 70, \ldots, 160\}$
instance_family	Graph family	complete_lognormal_iid
weight_range	Edge-weight interval	$(0, 1]$
σ	Lognormal shape parameter	2.0
quantile_clip	Weight clipping quantile	0.995
solver_model_pairs	Combinations of models and solvers benchmarked	(GRB,MCMIP), (GRBQ, MCQUBO), (SA, MCQUBO), (DW, MCQUBO)
num_problems_per_size	Instances generated per size	2
num_instances	Total instances in library	24
random_seed	Python RNG seed	42
num_reruns_qubo	Reruns for stochastic QUBO-based solver calls	3
GRB time_limit_s	Time limit for native MIP solve in seconds	20
GRBQ time_limit_s	Time limit for QUBO solve in seconds	20
SA num_reads	Number of annealing reads	100
SA num_sweeps	Number of sweeps per read	50
DW num_reads	Number of quantum reads	100
DW sampler	Quantum backend	Advantage_system6.4
metric	Optimization metric	cut value

Table 5: Configuration of the Maximum Cut case study.

The most notable observation is that, unlike in the vehicle routing case study, Gurobi performs best here on the quadratic model. Under the chosen 20-second time budget, (GRBQ, MCQUBO) achieves the strongest overall quality-versus-runtime trade-off across the entire tested range. At $N = 50$, (GRBQ, MCQUBO) already slightly improves on (GRB, MCMIP) while requiring only 0.36 seconds on average compared to 20.22 seconds for the native MIP baseline. The gap widens substantially with size: at $N = 100$, the mean cut value is 124.61 for (GRBQ, MCQUBO) versus 118.08 for (GRB, MCMIP), and at $N = 160$ the corresponding values are 286.21 versus 230.86. Hence, under the present benchmark budget, (GRBQ, MCQUBO) is the strongest classical baseline throughout the study.

Because raw Gurobi runtimes become less informative once the time limit is active, the lower panel of Figure 5 does not report them directly. Instead, for each instance and for each Gurobi-based model, we extract from the Gurobi log the first time at which the solver reaches the best cut value observed for D-Wave on that same instance. If the target value is not reached within the run, the corresponding entry is set to -1. This yields a direct classical time-to-target comparison against the D-Wave reference values.

Simulated annealing remains highly competitive in solution quality; across the larger sizes, its mean approximation ratio to the best observed solution stays close to

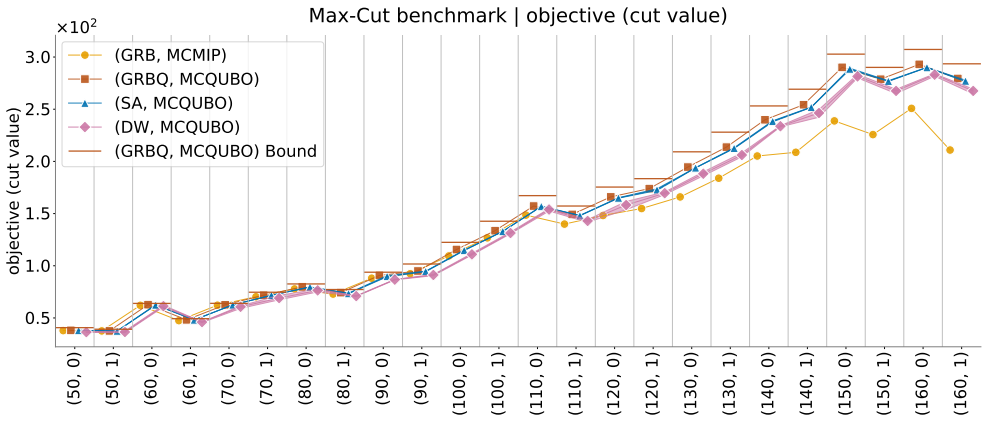


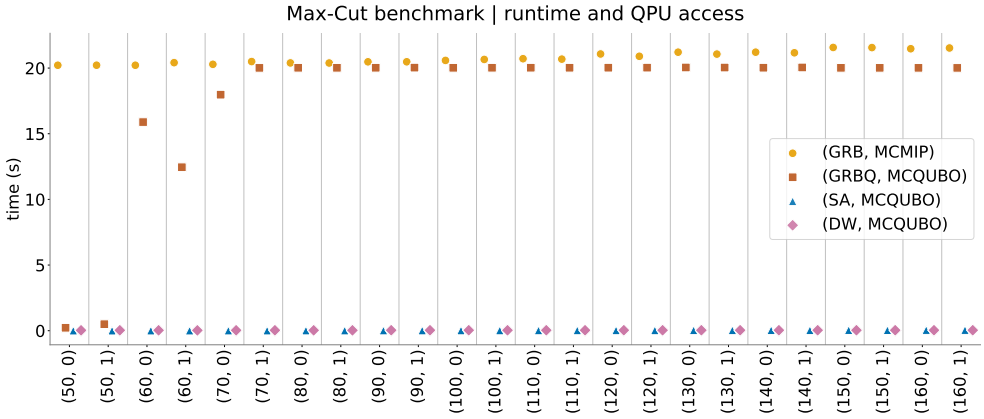
Fig. 4: Mean cut values for the Maximum-Cut case study across all tested graph sizes and solver-model combinations, where higher values are better. The instances are separated by gray vertical bars and each instance is noted as (N, i), where N is the number of nodes and i the index of the instance of this size. For repeated stochastic or quantum runs, markers denote mean values and shaded bands indicate the observed min–max range. Best observed bounds for (GRBQ, MCQUBO) are shown as horizontal lines in the corresponding color.

one, while its runtime remains below 0.05 seconds even at the largest tested size. Nevertheless, it is consistently slightly weaker than GRBQ and therefore serves primarily as a lightweight stochastic baseline rather than as the strongest overall method.

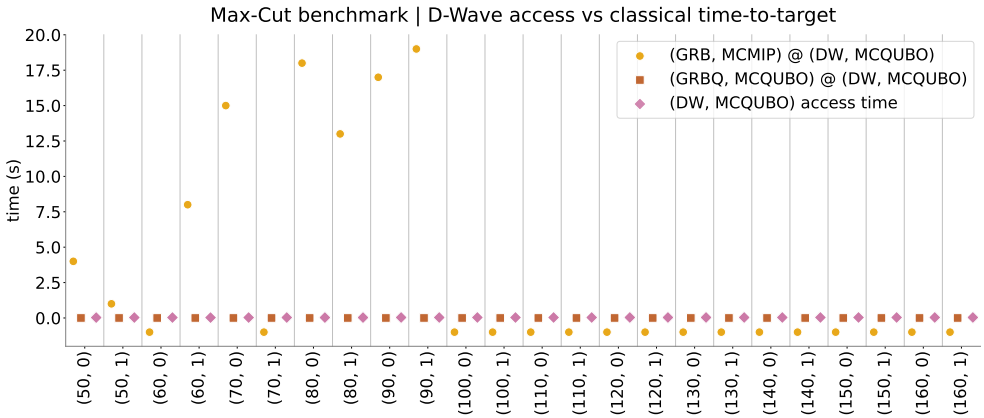
D-Wave Quantum yields the weakest solution quality among the four compared approaches on larger instances, although it remains consistently feasible. Its mean relative gap to the best observed solution stays in the low single-digit percentage range for the larger graphs, while the reported QPU access time remains almost constant at roughly 0.03 to 0.04 seconds across the tested size range. In contrast, the total end-to-end runtime grows substantially and reaches about 469.80 seconds on average at $N = 160$. This distinction is important for interpretation: the raw quantum hardware access remains short, but the practical workflow cost is dominated by additional overhead outside pure QPU time.

5 Conclusion and Outlook

We presented a use-case-oriented benchmark framework for comparing classical, quantum, hybrid, and quantum-inspired optimization methods across three dimensions: use cases, models, and solvers. The central design choice of the framework is to persist problem instances together with their derived model artifacts in libraries and to benchmark solver–model pairs on these libraries under a common execution and logging interface. This makes benchmark studies reproducible, interruptible, and incrementally extensible, while keeping instance generation, model construction, solver execution, and result analysis conceptually distinct.



(a) DW access time against wall-clock runtimes of GRB, GRBQ and SA.



(b) Classical time-to-target: for each instance, the first Gurobi time at which the best cut value observed for D-Wave on that instance is reached in comparison with the DW access time; if the target is not reached, the recorded value is -1.

Fig. 5: Timing-focused view of the Maximum-Cut case study. The lower panel compares D-Wave against Gurobi via a time-to-target metric and the DW access time.

The two case studies illustrate why this separation matters. In the vehicle routing study, solver performance depends strongly on model structure: Gurobi performs best on the native linear MIP model, whereas the Leap Hybrid solver group benefits from the binary models. In the Maximum-Cut study, the picture changes: under the chosen runtime budget, the quadratic model is particularly favorable for Gurobi, while simulated annealing remains competitive and D-Wave Quantum provides a hardware-oriented baseline with small QPU access time but noticeable end-to-end overhead. Overall, these results show that benchmark conclusions cannot be attributed to solver

technology alone; they depend on the interaction between solver and model, and this interaction is use-case-dependent.

This suggests several directions for future work. On the benchmark side, the framework can be extended by additional use cases, alternative models, and further backends, including stronger decomposition-based or variational methods. On the evaluation side, larger instance libraries, broader runtime budgets, and more systematic statistical aggregation across repeated runs would strengthen cross-technology comparisons. More generally, we view the framework not only as a collection of benchmark scripts, but as an extensible experimental protocol for studying when native models, penalty-based encodings, or hybrid constrained models are most appropriate for practical discrete optimization problems.

Supplementary information. Appendix A contains the full parameter lists and mathematical model descriptions for the two use cases studied in detail in the main text, namely Maximum Cut and Vehicle Routing. Detailed formulations for Maximum Independent Set, Container Scheduling, Portfolio Optimization and the Vehicle Routing CQM models are provided in the separate supplementary document.

Code availability. The benchmark framework implementation accompanying this work is publicly available on GitHub and archived on Zenodo [12]. The archived release corresponds to the public research-code release of the repository and is distributed under the Apache License 2.0.

Acknowledgements. This work was funded by the Bundesministerium für Forschung, Technologie und Raumfahrt (BMFTR) in the context of the Rymax One Project [grant number 13N16140]; the authors are grateful for the support. Furthermore, we thank Sven Jäger for his contributions to the framework's implementation and Raoul Heese for his initial conceptual input. Furthermore we like to thank Paolo Gunnellini, Nina Wagner, Marcel Morisse, Thomas Hübner and Volker Windeck for their valuable input for conceptualization of the use cases.

Declarations

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Appendix A Case-Study Use Cases and Models

This appendix records the full parameter sets and core formulations for the two use cases studied in the main text.

A.1 Maximum Cut

Given a simple, undirected graph $G = (V, E)$, $n := |V|$, with edge weights $W : E \to \mathbb{R}$, the cut value of $S \subseteq V$ is $\mathrm{cut}_W(S) = \sum_{\{i,j\}\in E,\, i\in S,\, j\notin S} W_{ij}$. The Maximum Cut problem is

$$\max_{S\subseteq V} \mathrm{cut}_W(S).$$

Throughout the following formulations, we use a symmetric weighted adjacency matrix $W \in \mathbb{R}^{n \times n}$ with $W_{ij} = W_{ji}$ and $W_{ii} = 0$, so every undirected edge weight is counted exactly once.

Parameter	Definition	Explanation
weighted_adjacency_matrix	symmetric matrix $W \in \mathbb{R}^{\lvert V \rvert \times \lvert V \rvert}$ with zero diagonal	Adjacency matrix of G with edge weights as entries

Table A1: MaximumCutProblem Parameters

A.1.1 MaximumCutQUBO (MCQUBO)

The MCQUBO reads

$$\min_{x \in \{0,1\}^n} E(x) := -\sum_{1 \le i < j \le n} W_{ij} \left(x_i \big(1 - x_j\big) + x_j \big(1 - x_i\big) \right).$$

where

$$x_i \in \{0,1\} \quad (i = 1, \ldots, n), \qquad x_i = \begin{cases} 1, & \text{if node } i \text{ is in } S, \\ 0, & \text{if node } i \text{ is in } V \setminus S. \end{cases}$$

A.1.2 MaximumCutMIP (MCMIP)

The MCMIP reads

$$\begin{aligned} \max \quad & \sum_{1 \le i < j \le n} W_{ij}\, y_{ij} \\ \text{s.t.} \quad & y_{ij} \le x_i + x_j && \forall 1 \le i < j \le n, \\ & y_{ij} \le 2 - x_i - x_j && \forall 1 \le i < j \le n, \\ & y_{ij} \ge x_i - x_j && \forall 1 \le i < j \le n, \\ & y_{ij} \ge x_j - x_i && \forall 1 \le i < j \le n, \end{aligned}$$

where

$$x_i \in \{0,1\} \quad (i = 1, \ldots, n), \quad x_i = \begin{cases} 1, & \text{if node } i \text{ is in } S, \\ 0, & \text{if node } i \text{ is in } V \setminus S. \end{cases}$$

$$y_{ij} \in \{0,1\} \quad (1 \le i < j \le n), \quad y_{ij} = \begin{cases} 1, & \text{if } x_i \ne x_j, \\ 0, & \text{otherwise.} \end{cases}$$

A.1.3 MaximumCutCQM (MCCQM)

The MCCQM is the constrained-quadratic counterpart of MCMIP. It uses the same binary partition variables and binary cut-edge indicators, but represents the formulation in the CQM interface. The MCCQM reads

$$\begin{aligned} \min \quad & -\sum_{1\le i<j\le n} W_{ij}\, y_{ij} \\ \text{s.t.} \quad & y_{ij} \le x_i + x_j && \forall 1 \le i < j \le n, \\ & y_{ij} + x_i + x_j \le 2 && \forall 1 \le i < j \le n, \\ & y_{ij} - x_i + x_j \ge 0 && \forall 1 \le i < j \le n, \\ & y_{ij} + x_i - x_j \ge 0 && \forall 1 \le i < j \le n, \end{aligned}$$

where

$$x_i \in \{0,1\} \quad (i = 1,\dots,n), \quad x_i = \begin{cases} 1, & \text{if node } i \text{ is in } S, \\ 0, & \text{if node } i \text{ is in } V \setminus S. \end{cases}$$

$$y_{ij} \in \{0,1\} \quad (1 \le i < j \le n), \quad y_{ij} = \begin{cases} 1, & \text{if nodes } i \text{ and } j \text{ are separated by the cut,} \\ 0, & \text{otherwise.} \end{cases}$$

A.2 Vehicle Routing

Let $V = \{0, 1, \dots, N-1\}$ be a set of locations with depot 0 and customers $V^\star = V \setminus \{0\}$. Let $D = (d_{uv})_{u,v\in V}$ be a nonnegative distance matrix. Given $M \in \mathbb{N}$ vehicles and a maximum number of customer stops $K \in \mathbb{N}$ per vehicle, the (stop-capacitated) vehicle routing problem is, find M routes R_j of the form

$$R_j = (0, r_{j,1}, r_{j,2}, \dots, r_{j,\ell_j}, 0), \qquad \ell_j \in \{1,\dots,K\}, \quad r_{j,t} \in V^\star,$$

such that every customer is visited exactly once across all vehicles, i.e.

$$\big\{ r_{j,t} \mid j = 0,\dots,M-1,\ t = 1,\dots,\ell_j \big\} = V^\star \quad \text{(disjointly)},$$

and the total route length is minimized:

$$\min \sum_{j=0}^{M-1} \left(d_{0,r_{j,1}} + \sum_{t=1}^{\ell_j - 1} d_{r_{j,t},r_{j,t+1}} + d_{r_{j,\ell_j},0} \right).$$

A.2.1 VehicleRoutingMIP (VRPMIP)

The VRPMIP reads

Parameter	Type	Explanation
distance_matrix	Matrix $D \in \mathbb{R}^{\lvert V\rvert \times \lvert V\rvert}$ with entries d_{uv}	Travel distance between nodes
num_vehicles	Integer $M \in \mathbb{N}_{>0}$	Number of vehicles M
vehicle_capacity	Integer $K \in \mathbb{N}_{>0}$	Vehicle capacity K, that is maximal number of destinations (nodes without hub) traversable

Table A2: VehicleRoutingProblem Parameters

$$\min_{x,u} \quad \sum_{u \in V} \sum_{\substack{v \in V \\ v \neq u}} d_{uv}\, x_{uv}$$

$$\text{s.t.} \quad \text{C1, C2, C3, C4}$$

where

$$x_{uv} \in \{0,1\} \quad (u, v = 0, \ldots, N-1; u \neq v), \quad x_{uv} = \begin{cases} 1, & \text{if } (u,v) \text{ in some route,} \\ 0, & \text{else.} \end{cases}$$

$$u_i \in \{0, \ldots, K-1\} \quad (i = 1, \ldots, N-1), \quad u_i = k, \text{ if } i \text{ at position } k \text{ in some route}$$

with the constraints

i) All destinations are left by exactly one vehicle:

$$\sum_{\substack{v \in V \\ v \neq i}} x_{iv} = 1 \quad \forall i \in V^\star, \tag{C1}$$

ii) All destinations are reached by exactly one vehicle:

$$\sum_{\substack{u \in V \\ u \neq j}} x_{uj} = 1 \quad \forall j \in V^\star, \tag{C2}$$

iii) Every transporter is used:

$$\sum_{v \in V^\star} x_{0v} = M, \tag{C3}$$

iv) MTZ subtour elimination:

$$u_i - u_j + K\, x_{ij} \leq K - 1 \quad \forall\, i, j \in V^\star,\ i \neq j. \tag{C4}$$

A.2.2 VehicleRoutingQUBO (VRPQUBO)

The VRPQUBO is a binary, position-indexed formulation of the stop-capacitated vehicle routing problem. Each variable $x_{i,j,k}$ indicates whether customer i is visited by vehicle j at route position k. The VRPQUBO reads

$$\min_{x\in\{0,1\}^{(N-1)MK}} E(x) = \lambda_1 P_1(x) + \lambda_2 P_2(x) + \lambda_3 P_3(x) + \lambda_4 P_4(x) + F(x),$$

where

$$x_{i,j,k} \in \{0,1\} \quad (i \in V^\star,\ j = 0,\dots,M-1,\ k = 0,\dots,K-1),$$

with

$$x_{i,j,k} = \begin{cases} 1, & \text{if customer } i \text{ is the } (k+1)\text{-th stop of vehicle } j, \\ 0, & \text{otherwise} \end{cases}$$

and the following penalty terms:

i) Each customer is served exactly once:

$$P_1(x) = \sum_{i\in V^\star} \left(\sum_{j=0}^{M-1} \sum_{k=0}^{K-1} x_{i,j,k} - 1 \right)^2. \tag{P_1}$$

ii) Each vehicle is at most at one customer per position k:

$$P_2(x) = \sum_{j=0}^{M-1} \sum_{k=0}^{K-1} \left(\sum_{i\in V^\star} x_{i,j,k} \right) \left(\sum_{i\in V^\star} x_{i,j,k} - 1 \right). \tag{P_2}$$

iii) If a vehicle has a $(k+2)$-th stop, then it must also have a $(k+1)$-th stop:

$$P_3(x) = \sum_{j=0}^{M-1} \sum_{k=0}^{K-2} \Delta_{j,k}(x)\big(\Delta_{j,k}(x) - 1\big), \qquad \Delta_{j,k}(x) = \sum_{i\in V^\star} \big(x_{i,j,k} - x_{i,j,k+1}\big). \tag{P_3}$$

iv) Each vehicle has exactly one first stop:

$$P_4(x) = \sum_{j=0}^{M-1} \left(\sum_{i\in V^\star} x_{i,j,0} - 1 \right)^2. \tag{P_4}$$

The objective function is the sum of route lengths of each vehicle

$$F(x) = F_{\text{first}}(x) + F_{\text{inner}}(x) + F_{\text{last}}(x), \tag{F}$$

where

$$F_{\text{first}}(x) = \sum_{j=0}^{M-1} \sum_{i \in V^\star} d_{0i}\, x_{i,j,0},$$

$$F_{\text{inner}}(x) = \sum_{j=0}^{M-1} \sum_{k=0}^{K-2} \sum_{u \in V^\star} \sum_{\substack{v \in V^\star \\ v \neq u}} d_{uv}\, x_{u,j,k}\, x_{v,j,k+1},$$

and

$$F_{\text{last}}(x) = \sum_{j=0}^{M-1} \left[\sum_{u \in V^\star} d_{u0}\, x_{u,j,K-1} + \sum_{k=0}^{K-2} \sum_{u \in V^\star} d_{u0}\, x_{u,j,k} \left(1 - \sum_{v \in V^\star} x_{v,j,k+1} \right) \right].$$

Penalties are scaled by the parameters λ, which are set by default as follows: Let $d_{\max} = \max_{u,v \in V} d_{uv}$, then

$$\lambda_1 = \lambda_2 = 2d_{\max} + 1, \qquad \lambda_3 = d_{\max} + 1, \qquad \lambda_4 = d_{\max}(N+1).$$

A.2.3 VehicleRoutingBinaryMIP (VRPMIPB)

The VRPMIPB is the corresponding binary, position-indexed mixed-integer formulation. It uses the same assignment variables and route-length expression as VRPQUBO, but replaces the penalty terms of the QUBO model by explicit hard constraints. The VRPMIPB reads

$$\min_{x \in \{0,1\}^{(N-1)MK}} F(x),$$
$$\text{s.t.} \quad \text{C1, C2, C3, C4}$$

where

$$x_{i,j,k} \in \{0,1\} \quad (i \in V^\star,\ j = 0, \ldots, M-1,\ k = 0, \ldots, K-1),$$

with

$$x_{i,j,k} = \begin{cases} 1, & \text{if customer } i \text{ is the } (k+1)\text{-th stop of vehicle } j, \\ 0, & \text{otherwise.} \end{cases}$$

This formulation is essentially the same position-indexed routing model as VRPQUBO, but the penalty terms are enforced here as hard constraints instead of being added to the objective. The objective function is again the sum of route lengths of each vehicle

$$F(x) = F_{\text{first}}(x) + F_{\text{inner}}(x) + F_{\text{last}}(x),$$

where

$$F_{\text{first}}(x) = \sum_{j=0}^{M-1} \sum_{i \in V^\star} d_{0i}\, x_{i,j,0},$$

$$F_{\text{inner}}(x) = \sum_{j=0}^{M-1} \sum_{k=0}^{K-2} \sum_{u \in V^\star} \sum_{\substack{v \in V^\star \\ v \neq u}} d_{uv}\, x_{u,j,k}\, x_{v,j,k+1},$$

and

$$F_{\text{last}}(x) = \sum_{j=0}^{M-1} \left[\sum_{u \in V^\star} d_{u0}\, x_{u,j,K-1} + \sum_{k=0}^{K-2} \sum_{u \in V^\star} d_{u0}\, x_{u,j,k} \left(1 - \sum_{v \in V^\star} x_{v,j,k+1} \right) \right],$$

with the constraints

i) Each customer is served exactly once:

$$\sum_{j=0}^{M-1} \sum_{k=0}^{K-1} x_{i,j,k} = 1 \quad \forall i \in V^\star. \tag{C1}$$

ii) Each vehicle is at most at one customer per position k:

$$\sum_{i \in V^\star} x_{i,j,k} \leq 1 \quad \forall j = 0, \ldots, M-1,\ k = 0, \ldots, K-1. \tag{C2}$$

iii) If a vehicle has a $(k+1)$-th stop, then it must also have a k-th stop:

$$\sum_{i \in V^\star} x_{i,j,k} - \sum_{i \in V^\star} x_{i,j,k-1} \leq 0 \quad \forall j = 0, \ldots, M-1,\ k = 1, \ldots, K-1. \tag{C3}$$

iv) Each vehicle has exactly one first stop:

$$\sum_{i \in V^\star} x_{i,j,0} = 1 \quad \forall j = 0, \ldots, M-1. \tag{C4}$$

A.2.4 VehicleRoutingCQM (VRPCQM) and VehicleRoutingBinaryCQM (VRPCQMB)

The two CQM variants correspond to the two MIP formulations in constrained-quadratic form. VRPCQM is the CQM counterpart of the native arc-based formulation VRPMIP, whereas VRPCQMB is the CQM counterpart of the binary position-indexed formulation VRPMIPB.

The native VRPCQM uses the same arc variables x_{uv} and ordering variables u_i as VRPMIP, together with the same routing objective and the same four constraint families from (C1)–(C4). In other words, the routing semantics are unchanged; only the model is represented in the CQM interface instead of the classical MIP interface.

Analogously, VRPCQMB uses the same binary position-indexed assignment variables $x_{i,j,k}$ and the same route-length objective $F(x)$ as VRPMIPB. Its constraints are exactly the binary-routing constraints from (C1)–(C4), but encoded directly in

the CQM model. Hence, VRPCQMB is the constrained-quadratic analogue of the position-indexed formulation underlying VRPQUBO and VRPMIPB.

References

[1] Jain, R.: The Art of Computer Systems Performance Analysis - Techniques for Experimental Design, Measurement, Simulation, and Modeling. Wiley professional computing, pp. –1685. Wiley, New York, NY (1991). https://doi.org/10.1093/comjnl/35.3.207

[2] Camp, R.C.: Benchmarking: The Search for Industry Best Practices that Lead to Superior Performance. Productivity Press, New York (2006). https://doi.org/10.4324/9781003578871 . Reprint of the 1989 original

[3] Bartz-Beielstein, T., Doerr, C., Berg, D., Bossek, J., Chandrasekaran, S., Eftimov, T., Fischbach, A., Kerschke, P., Cava, W.L., Lopez-Ibanez, M., Malan, K.M., Moore, J.H., Naujoks, B., Orzechowski, P., Volz, V., Wagner, M., Weise, T.: Benchmarking in Optimization: Best Practice and Open Issues (2020). https://arxiv.org/abs/2007.03488

[4] Proctor, T., Young, K., Baczewski, A.D., Blume-Kohout, R.: Benchmarking quantum computers. Nature Reviews Physics **7**(2), 105–118 (2025) https://doi.org/10.1038/s42254-024-00796-z

[5] Acuaviva, A., Aguirre, D., Peña, R., Sanz, M.: Benchmarking quantum computers: towards a standard performance evaluation approach. Quantum Science and Technology **11** (2026) https://doi.org/10.1088/2058-9565/ae4269

[6] Resch, S., Karpuzcu, U.R.: Benchmarking quantum computers and the impact of quantum noise. ACM Comput. Surv. **54**(7) (2021) https://doi.org/10.1145/3464420

[7] Martiel, S., Ayral, T., Allouche, C.: Benchmarking quantum coprocessors in an application-centric, hardware-agnostic, and scalable way. IEEE Transactions on Quantum Engineering **PP**, 1–1 (2021) https://doi.org/10.1109/TQE.2021.3090207

[8] Granet, E., Dreyer, H.: Appqsim: Application-oriented benchmarks for hamiltonian simulation on a quantum computer. Physical Review Research **7**(4) (2025) https://doi.org/10.1103/qt55-7d6r

[9] Tomesh, T., Gokhale, P., Omole, V., Ravi, G.S., Smith, K.N., Viszlai, J., Wu, X.-C., Hardavellas, N., Martonosi, M.R., Chong, F.T.: SupermarQ: A Scalable Quantum Benchmark Suite (2022). https://arxiv.org/abs/2202.11045

[10] Finzgar, J.R., Ross, P., Holscher, L., Klepsch, J., Luckow, A.: Quark: A framework for quantum computing application benchmarking. In: 2022 IEEE International

Conference on Quantum Computing and Engineering (QCE), pp. 226–237. IEEE, Broomfield, CO, USA (2022). https://doi.org/10.1109/qce53715.2022.00042

[11] Koch, T., Neira, D.E.B., Chen, Y., Cortiana, G., Egger, D.J., Heese, R., Hegade, N.N., Cadavid, A.G., Huang, R., Itoko, T., Kleinert, T., Xavier, P.M., Mohseni, N., Montanez-Barrera, J.A., Nakano, K., Nannicini, G., O'Meara, C., Pauckert, J., Proissl, M., Ramesh, A., Schicker, M., Shimada, N., Takeori, M., Valls, V., Bulck, D.V., Woerner, S., Zoufal, C.: Quantum Optimization Benchmarking Library - The Intractable Decathlon (2025). https://arxiv.org/abs/2504.03832

[12] Leib, D.: qubo-library-public. Zenodo (2026). https://doi.org/10.5281/zenodo.20397253 . https://doi.org/10.5281/zenodo.20397253

[13] Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness (Series of Books in the Mathematical Sciences), First edition edn. W. H. Freeman, San Francisco, CA (1979)

[14] Goemans, M.X., Williamson, D.P.: Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. J. ACM **42**(6), 1115–1145 (1995) https://doi.org/10.1145/227683.227684

[15] Korte, B.H., Vygen, J.: Combinatorial Optimization: Theory and Algorithms. Springer, New York, NY (2012). https://doi.org/10.1007/978-3-642-24488-9

[16] Toth, P., Vigo, D., Industrial, S., Mathematics., A.: The Vehicle Routing Problem. SIAM monographs on discrete mathematics and applications. Society for Industrial and Applied Mathematics SIAM, 3600 Market Street, Floor 6, Philadelphia, PA 19104, Philadelphia, Pa (2001)

[17] Laporte, G.: Fifty years of vehicle routing. Transportation Science **43**(4), 408–416 (2009) https://doi.org/10.1287/trsc.1090.0301

[18] Steenken, D., Voß, S., Stahlbock, R.: Container terminal operation and operations research - a classification and literature review. OR Spectrum **26**, 3–49 (2004) https://doi.org/10.1007/s00291-003-0157-z

[19] Pinedo, M.L.: Scheduling: Theory, Algorithms, and Systems, 3rd edn. Springer, New York, NY (2008)

[20] Markowitz, H.: Portfolio selection. The Journal of Finance **7**(1), 77–91 (1952) https://doi.org/10.1111/j.1540-6261.1952.tb01525.x https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1540-6261.1952.tb01525.x

[21] Kellerer, H., Pferschy, U., Pisinger, D.: Knapsack Problems. Springer, Berlin (2004)

[22] Nemhauser, G.L., Wolsey, L.A.: Integer and Combinatorial Optimization. John

Wiley & Sons, New York, NY (1988). https://doi.org/10.1112/blms/21.5.499

[23] Bertsimas, D., Weismantel, R.: Optimization over Integers, pp. –1602. Athena Scientific, Belmont, MA (2005)

[24] Kochenberger, G., Hao, J.-K., Glover, F., Lewis, M., Lü, Z., Wang, H., Wang, Y.: The unconstrained binary quadratic programming problem: a survey. Journal of Combinatorial Optimization **28**, 58–81 (2014) https://doi.org/10.1007/s10878-014-9734-0

[25] Glover, F., Kochenberger, G.A., Du, Y.: Quantum bridge analytics i: A tutorial on formulating and using qubo models. 4OR **17**, 335–371 (2019) https://doi.org/10.1007/s10479-022-04634-2

[26] D-Wave Systems Inc.: Ocean Software Documentation: Constrained Quadratic Model (CQM). https://docs.ocean.dwavesys.com/

[27] D-Wave Systems Inc.: Leap hybrid constrained quadratic model solver. Technical report, D-Wave Systems Inc. (2021). White paper

[28] Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P.: Optimization by simulated annealing. Science **220**(4598), 671–680 (1983) https://doi.org/10.1126/science.220.4598.671

[29] Aarts, E. (ed.): Simulated Annealing & Boltzmann Machines: a Stochastic Approach to Combinatorialoptimization & Neural Computing. John Wiley & Sons, Inc., USA (1988). https://doi.org/10.2307/2008816

[30] Kadowaki, T., Nishimori, H.: Quantum annealing in the transverse ising model. Phys. Rev. E **58**, 5355–5363 (1998) https://doi.org/10.1103/PhysRevE.58.5355

[31] Johnson, M.W., Amin, M.H.S., Gildert, S., Lanting, T., Hamze, F., Dickson, N., Harris, R., Berkley, A.J., Johansson, J., Bunyk, P., Chapple, E.M., Enderud, C., Hilton, J.P., Karimi, K., Ladizinsky, E., Ladizinsky, N., Oh, T., Perminov, I., Rich, C., Thom, M.C., Tolkacheva, E., Truncik, C.J.S., Uchaikin, S., Wang, J., Wilson, B., Rose, G.: Quantum annealing with manufactured spins. Nature **473**(7346), 194–198 (2011) https://doi.org/10.1038/nature10012

[32] D-Wave Systems Inc.: D-Wave Hybrid Solver Service: An Overview. https://www.dwavesys.com/media/4bnpi53x/14-1039a-b_d-wave_hybrid_solver_service_an_overview.pdf (2020)

[33] Michael Booth, Steven P. Reinhardt and Aidan Roy: Partitioning Optimization Problems for Hybrid Classical/Quantum Execution. https://docs.ocean.dwavesys.com/projects/qbsolv/en/latest/_downloads/bd15a2d8f32e587e9e5997ce9d5512cc/qbsolv_techReport.pdf (2017)

[34] Farhi, E., Goldstone, J., Gutmann, S.: A Quantum Approximate Optimization

Algorithm (2014). https://arxiv.org/abs/1411.4028

[35] Miller, C.E., Tucker, A.W., Zemlin, R.A.: Integer programming formulation of traveling salesman problems. J. ACM **7**(4), 326–329 (1960) https://doi.org/10.1145/321043.321046

[36] Deza, M.M., Laurent, M.: Geometry of Cuts and Metrics. Algorithms and combinatorics, vol. 15. Springer, Germany (1997). Pagination: XII, 587

[37] Lucas, A.: Ising formulations of many NP problems. Frontiers in Physics **2** (2014) https://doi.org/10.3389/fphy.2014.00005

Supplementary Files

This is a list of supplementary files associated with this preprint. Click to download.

- supplementary.pdf