Quantum Optimization Benchmarking Library
The Intractable Decathlon

Thorsten Koch *[1, 2], David E. Bernal Neira[3], Ying Chen[4], Giorgio Cortiana[5], Daniel J. Egger[6], Raoul Heese[7], Narendra N. Hegade[8], Alejandro Gomez Cadavid[8,9], Rhea Huang[10], Toshinari Itoko[11], Thomas Kleinert[12], Pedro Maciel Xavier[3, 13], Naeimeh Mohseni[5], Jhon A. Montanez-Barrera[14], Koji Nakano[15], Giacomo Nannicini[10], Corey O'Meara[5], Justin Pauckert[16], Manuel Proissl[6], Anurag Ramesh[3], Maximilian Schicker[1], Noriaki Shimada[11], Mitsuharu Takeori[11], Víctor Valls[17], David Van Bulck[18], Stefan Woerner †[6], and Christa Zoufal ‡[6]

[1] *Zuse Institute Berlin*
[2] *Technische Universität Berlin*
[3] *Davidson School of Chemical Engineering, Purdue University*
[4] *National University of Singapore*
[5] *E.ON Digital Technology GmbH*
[6] *IBM Quantum, IBM Research Europe – Zurich*
[7] *NTT Data*
[8] *Kipu Quantum GmbH*
[9] *University of the Basque Country UPV/EHU*
[10] *University of Southern California*
[11] *IBM Quantum, IBM Research Tokyo*
[12] *Quantagonia*
[13] *Federal University of Rio de Janeiro*
[14] *Forschungszentrum Jülich*
[15] *Hiroshima University*
[16] *T-Systems International GmbH*
[17] *IBM Quantum, IBM Research Europe – Dublin*
[18] *Ghent University*

August 29, 2025

Abstract

Through recent progress in hardware development, quantum computers have advanced to the point where *benchmarking* of (heuristic) quantum algorithms at scale is within reach. Particularly in combinatorial optimization–where most algorithms are heuristics–it is key to empirically analyze their performance on hardware and track progress towards quantum advantage. To this extent, we present *ten optimization problem classes* that are difficult for existing classical algorithms and can (mostly) be linked to practically relevant applications, with the goal to enable systematic, fair, and comparable benchmarks for quantum optimization methods. Further, we introduce the *Quantum Optimization Benchmarking Library* [QOBLIB] where the problem instances and solution track records can be found. The individual properties of the problem classes vary in terms of objective and variable type, coefficient ranges, and density. Crucially, they all become challenging for established classical methods already at system sizes ranging from less than 100 to, at most, an order of 100 000 decision variables, allowing to approach them with today's quantum computers. We reference the results from state-of-the-art solvers for instances from all problem classes and demonstrate exemplary baseline results obtained with quantum solvers for selected problems. The baseline results illustrate a standardized form to present benchmarking solutions, which has been designed to ensure *comparability* of the used methods, *reproducibility* of the respective results, and *trackability* of algorithmic and hardware improvements over time. We encourage the optimization community to explore the performance of available classical or quantum algorithms and hardware platforms with the benchmarking problem instances presented in this work toward demonstrating quantum advantage in optimization.

*koch@zib.de
†wor@zurich.ibm.com
‡ouf@zurich.ibm.com

Contents

1 Introduction

Optimization is subject to extensive research due to its importance in science and industry. Applications include experiment design, protein folding, logistics, and finance, where better algorithms can improve performance, accuracy, and cost efficiency. Optimization algorithms fall into three categories: provably exact algorithms, provably approximate algorithms with a priori performance guarantees, and heuristic algorithms without such guarantees, although posterior performance bounds may sometimes be available. Complexity theory classifies optimization problems as "easy" or "difficult", determining what types of algorithms may exist. Many relevant optimization problems are **NP**-hard, i.e., the computational resources required to find provably optimal solutions grow exponentially with the problem size. We do not expect the existence of provably exact efficient algorithms for such problems in general [AAA+24]. For some of them, the possible performance of provably approximate algorithms is limited by known inapproximability bounds cf. [GW95, Kho02, KKMO07]. Here, inapproximability bounds imply that solving a problem better than the given bound is **NP**-hard, i.e., as hard as solving the problem to optimality. In practice, it is often sufficient to find a good solution instead of a provably optimal one. Thus, classic heuristics and ML-based solvers are viable approaches for many problems of interest. Since heuristics lack a priori guarantees, rigorous benchmarking is crucial to assess the corresponding performance and resource requirements. To enable a fair comparison of algorithms and hardware platforms and to guarantee reproducibility of reported results, a suitable set of problem instances and metrics needs to be defined. This also allows tracking progress in algorithm and hardware improvements over time.

Quantum computing offers a novel toolbox to approach optimization problems–see [AAA+24] for an in-depth discussion of quantum optimization algorithms, their challenges, and potential. With recent advances in quantum hardware [KEA+23], systematic benchmarking of quantum optimization algorithms and comparison with classical solvers have become feasible. This paper proposes ten classes of combinatorial optimization problems, see Table 1, and corresponding problem instances, which can be found in an open-source repository *Quantum Optimization Benchmark Library* [QOBLIB]. These problem classes were selected because they include instances that are difficult for state-of-the-art classical solvers with only a hundred to a hundred thousand variables, have diverse characteristics, and many of them have connections to industrial applications. This makes them suitable candidates for benchmarking and exploring the potential of quantum advantage in optimization. Further, for each class, we provide instances of varying complexity, ranging from those already feasible for today's quantum hardware to those challenging for state-of-the-art classical solvers.

Problem	Description
Market Split	Multi-dimensional subset-sum
LABS	Low autocorrelation binary sequences
Birkhoff	Minimum Birkhoff decomposition
Steiner	Steiner tree packing in graphs (VLSI Design/Wire Routing)
Sports	Sports tournament scheduling (STS)
Portfolio	Multi-period portfolio optimization with transaction costs
Independent Set	Unweighted maximum independent set (MIS)
Network	Communications network design problem
Routing	Capacitated vehicle routing problem (CVRP)
Topology	Graph topology design (Node-Degree-Diameter problem)

Table 1: The intractable decathlon: Ten combinatorial optimization and feasibility problem classes that are difficult to solve with state-of-the-art solvers and can already be studied with today's quantum computers. They are introduced in detail in Section 4.

Notably, we promote model-independent benchmarking, which allows for maximal flexibility in choosing models, algorithms, and hardware platforms. A prerequisite for our goal is the establishment of a fair and systematic benchmarking effort to advance toward demonstrating quantum advantage in optimization. Given the large number of possibilities, exhaustively comparing algorithms is an enormous challenge that requires the scientific community's collective effort.

In addition to identifying the best approach to solving a given problem, benchmarking is critical for verifying algorithms' correctness, identifying bottlenecks, improving existing approaches, and tracking progress over time. This includes the progress in algorithms, software libraries, and hardware platforms [NSB+24]. The goal, scope, and limitations of a benchmarking effort must be clearly defined, as this determines the type of conclusions it may support. An overview of the main goals of benchmarking is presented below in the box "Goals in Benchmarking", cf. [FRH+22] for a related discussion.

The remainder of this work is structured as follows. First, we discuss related work focusing on quantum application benchmarking in Section 2. Then, in Section 3, we present our benchmarking methodology, covering different modeling approaches and key metrics to ensure reproducibility, comparability, and trackability of results. Afterward, Section 4 introduces the ten benchmarking problem classes. In addition to defining the problem, this section also provides additional background, such as explaining the relevance of these problems for quantum optimization benchmarking and presenting classical baseline results. To kick off this collaborative benchmarking effort, Section 5 discusses how to contribute solutions to the open-source repository and provides illustrative quantum baseline results for selected problem instances. We conclude in Section 6.

Goals in Benchmarking

Applications Benchmarking: The goal is to find the best possible algorithms–classical or quantum–to solve a given problem instance. Thus, benchmarks must be model-independent to allow all possible approaches to solve a problem. This is the only level that ultimately allows demonstrating quantum advantage.

Algorithm Benchmarking: The goal is to identify suitable strategies for setting hyperparameters, identifying bottlenecks, improving algorithms, and tracking progress over time. Since algorithm benchmarking does not entail comparing against all possible algorithms, it will not allow demonstrating quantum advantage. Nevertheless, it can be used to estimate an algorithm's scaling, which may facilitate the identification of potential asymptotic scaling advantages and help track progress toward quantum advantage.

System Benchmarking: The goal is to identify the best way to run a fixed algorithm for a fixed problem on a given platform. This includes tuning algorithmic hyperparameters or parameters of the execution environment, e.g., for error suppression and mitigation, and to confirm that the algorithm is working as expected. It can also be used for application-centric hardware benchmarking.

2 Related Work

A general introduction to benchmarking can be found in [DGS18, Jai91, KBPV22, BBDvdB+20, MWHK11, DMBS79]. Many important classical optimization benchmarks have been established via competitions and open-source libraries. Notable examples are the Discrete Mathematics and Theoretical Computer Science (DIMACS) implementation challenges [dim], the Satisfiability (SAT) competitions [FHI+21, SAT], and the Mixed Integer Programming Library (MIPLIB) [GHG+21]. Another set of optimization benchmarks is regularly conducted by Hans Mittelmann [Mit23]. Here, different optimization pipelines are executed using the same computational setup.

Most benchmarks in quantum computing focus on the performance of individual components of the quantum computing stack and their integration [PYBBK24, AAPS24, LME+25]: Randomized benchmarking focuses on gate quality [MGJ+12]. The layer fidelity represents a quality metric of layers of two-qubit gates [MHP+23]. Circuit Layer Operations Per Second (CLOPS) measures the speed of a quantum computer [NSB+24, WPJA+21]. Speed and quality of compiling quantum circuits from high-level abstract representations to machine-level instructions have been investigated and reported in *Benchpress* [NSB+24]. In addition, application-driven benchmarks have been proposed as full-stack system benchmarks. They usually focus on problems that can be solved to optimality classically and track the progress of quantum computers to achieve a known optimal solution [GD25, MAA21, LCM+24, TGO+22, SJHE24, FRH+22]. While this allows tracking the progress of hardware and algorithms, it is not sufficient to demonstrate quantum advantage.

This work presents a selected set of optimization problem instances that are difficult for state-of-the-art algorithms and corresponding benchmarking metrics that enable a fair and reliable comparison. See [McG24b, BBS+24, LAZ+25] for further discussion on the importance of appropriately chosen benchmarking metrics. The presented set of problem instances enables investigating potential advantages of novel algorithms and computational platforms. Furthermore, it facilitates tracking the progress towards potential quantum advantage in optimization. This is comparable to the curated set of many-body quantum systems presented in [WRV+24]–but for optimization.

3 Methods

In this section, we introduce our methodology for quantum optimization benchmarking. More specifically, we discuss what general problem types are considered, different ways to model them, and metrics to measure the performance of a specific optimization workflow.

3.1 Problem Types

We consider ten classes of combinatorial optimization problems. Schrijver [Sch03] describes combinatorial optimization as follows: *"Combinatorial Optimization searches for an optimum object in a finite collection of objects. Typically, the collection has a concise representation, while the number of objects is huge–more precisely, it grows exponentially in the size of the representation. So scanning all objects one by one and selecting the best one is not an option"*. There are other classes of optimization problems for which quantum solvers have been proposed, including several classes of polynomial-time solvable convex optimization problems; see [AAA+24] for an overview. We focus on combinatorial optimization for the following two reasons: First, quantum algorithms for convex optimization usually require fault-tolerant quantum computers. Hence, they cannot be tested on existing quantum devices. For combinatorial optimization, on the other hand, multiple (heuristic) approaches are known that can be implemented and tested already today. Second, classical computers excel at solving convex optimization problems even at massive scales, while combinatorial optimization problems can be intractable for classical algorithms already at a relatively small number of variables. This increases the possibility that near-term quantum computers outperform classical ones in certain cases.

More specifically, we consider combinatorial optimization problems with $n \in \mathbb{N}$ decision variables of the form

$$x^* = \operatorname*{argmin}_{x \in X} f(x),$$

with feasible space $X = \{x \in \mathbb{Z}^n \mid g(x) \leq b, \, l \leq x \leq u\}$, where $g : \mathbb{Z}^n \to \mathbb{Z}$, $b \in \mathbb{Z}, l, u \in \mathbb{Z}^n$, and objective function $f : X \to \mathbb{Z}$. It should be noted that the coefficients are (if necessary) rescaled and rounded to integers to mitigate potential numerical accuracy issues and to simplify the solution. We distinguish between *feasibility problems* and *optimization problems*. For feasibility problems, we can assume f is constant, and we are interested in finding any solution $x^* \in X$ that satisfies all given constraints, or in showing that $X = \emptyset$. For optimization problems, we have a non-constant objective function to be minimized or maximized, in addition to potential constraints to be satisfied. If an algorithm returns a solution $\hat{x} \in X$, we call the solution *feasible*. If in addition, $f(\hat{x}) = \min_{x \in X} f(x)$, we call the solution *optimal*. Multiple optimal solutions may exist.

From the theoretical side, both problem types are closely related, as we can solve optimization problems via a sequence of feasibility problems by adding a constraint for a certain objective value k, i.e., $X_k = \{x \in X \mid f(x) \leq k\}$, and asking whether $X_k = \emptyset$. The value of k can then be iteratively adjusted, for example, using binary search, until the optimal value is found. We can also make the problem unconstrained, for instance, by removing the constraint and instead minimizing $\bar{f}(x) = f(x) + M \cdot \max\{0, g(x) - b\}$, $X = \{x \in \mathbb{Z}^n \mid l \leq x \leq u\}$ for a suitably large M, see Section 3.2 for a more detailed discussion.

In practice, a good solution without proof of optimality or provable bound on the approximation ratio is often sufficient. If a formal guarantee on the solution quality is not required, finding a feasible solution with a good objective value is often achievable with classical heuristics, although challenging problem instances exist. We restrict our benchmarking study to computing good solutions without requiring a proof of optimality. First, because we want to assess practical performance, and second, because most known quantum optimization algorithms are heuristics and do not provide any a priori or a posteriori performance guarantees.

3.2 Modeling

Mathematical programming became a fundamental tool in many disciplines thanks to the existence of high-performance solvers that can find good–often even optimal–solutions for large classes of problems. Mathematical programming provides a way to classify models according to their structure and match model families with their respective specialized algorithms. The invention of more advanced solvers led to a shift in how problems are modeled and solved. As an example, the impactful success of solvers for *Mixed Integer Programming* (MIP) resulted in many applications that were reformulated into linear formulations to benefit from highly performant implementations. Modern MIP solvers can solve–often to proven optimality–Mixed Integer Linear and Quadratic

Programs (MILP/MIQP), as well as their pure Integer and Binary variants, ILP, IQP, BLP, and BQP, respectively.

The choice of problem formulation, even in the same model class, can have a tremendous impact on the solvability of a problem [AKT08]. While the global optimum is typically the same for different problem formulations[1], one might be better suited for a particular problem instance or algorithm than another. The development of novel hardware and algorithmic devices to solve *Quadratic Unconstrained Binary Optimization* (QUBO) problems led to a renewed interest in *formulations* of problems as QUBO [Luc14, GKD18]. While all *Integer Linear Programs* (ILP), or *Integer Quadratic Programs* (IQP) can be arbitrarily approximated by some QUBO, naive reformulations will often contain a large number of densely connected variables and will require outstanding precision for the coefficients to accurately represent the original structure [ARCR+24]. For example, ILP can explicitly represent different constraints or variables that can be difficult to encode into a QUBO, since many constraints can only be incorporated implicitly into QUBO via penalty terms that vanish for feasible solutions. In addition, to realize a QUBO representation, integer variables must be decomposed into binary ones–a useful library to handle this conversion can be found at [QUB25]. Direct translation from a general ILP to a QUBO requires making decisions on hyperparameters that define the encoding and penalty functions [GKD18] and can impact the quality of a solution depending on the considered algorithm. Furthermore, comparing the results received from multiple formulations of a problem requires careful interpretation. The objective values of two QUBOs representing the same problem using different penalty factors cannot be directly compared. Therefore, solutions must be mapped back to the original problem to understand and compare their performance. Higher-level modeling languages exist that provide additional structure to accommodate a problem's original semantics more naturally, and multiple tools can automate reformulating from a high-level representation into, for instance, ILP or QUBO; see, e.g., [Koc04, XRA+23, VQG+24]. Thus, the fact that a problem can be cast into a certain form does not necessarily mean that, in practice, it is a good idea to solve it in that form. Given the inherent formulation differences, we focus on *model-independent benchmarking* such that one may find the best formulation and corresponding algorithm for a given problem instance. Another type of reformulation allows mapping *Higher-Order Unconstrained Binary Optimization* (HUBOs), which may either be solved directly [PBE24] or converted to QUBOs [ABCG17, I75]. To this extent, higher-order binary terms are broken into multiple quadratic terms by introducing additional binary variables. Depending on the number and degree of the higher-order terms, this can significantly increase the number of binary variables, potentially leading to impractical problem formulations. Some quantum optimization algorithms can handle higher-order terms natively, giving them a potential advantage over those algorithms (quantum and classical) that cannot. Notably, many quantum optimization algorithms are designed for QUBOs. A common approach is, thus, to start with a higher-level problem formulation and reformulate it as QUBO. Tutorials on QUBO reformulations can be found, e.g., in [GKD18, XRP+24].

3.3 Metrics

The different possible goals of running a benchmark, as described in Section 1, require different metrics to be considered. To ensure that benchmarking results allow for a fair comparison, the reported metrics must be clearly defined. In the following, we will focus on *solution quality* and required *computational resources* and discuss the corresponding metrics. Examples of how to report benchmarking results for selected problem instances and quantum algorithms are presented in Section 5.

3.3.1 Solution Quality

The metrics for evaluating solution quality differ between feasibility and optimization problems. A feasibility problem is completely solved as soon as a feasible solution is found. However, proving that no such solution exists is usually significantly more challenging and impossible for most heuristics. For optimization problems, the key metric to be reported is the *best objective value* corresponding to a *feasible solution* found by an algorithm. If the algorithm also provides a lower bound (for minimization) or an upper bound (for maximization) for the optimal objective value, this bound can be provided as well. To solve an optimization problem to *proven optimality*, an algorithm must first find an optimal $x^* \in X$ and then prove that no better $x \in X$ exists with $f(x) < f(x^*)$, e.g., by providing a tight bound on the objective value. Usually, the second step is the more difficult one.

[1]Variations may occur, for example, in relaxed, e.g., linear or semi-definite programming formulations.

The above respective algorithms may run deterministically or, as many classical and most quantum algorithms do, stochastically. In the latter case, additional details should be reported. Stochastic algorithms should be executed multiple times to ensure the reported results describe the typical behavior of the algorithm. Then, for both feasibility and optimization problems, the number of repetitions yielding feasible solutions should be reported. For optimization problems, additionally, the best objective value achieved by a feasible solution found across all repetitions should be stated, as well as the number of *successful* repetitions. Given a number of repetitions of a stochastic algorithm and a given tolerance $\epsilon \geq 0$, we consider a single run successful if it finds a feasible solution with an objective value that is ϵ-close to the best-found feasible solution across all repetitions. More precisely, suppose the best found solution over all repetitions, denoted by x' with corresponding objective value $f(x')$, which may differ from the global optimum $f(x^*)$. Then, a solution x corresponding to a successful run must satisfy $f(x) \leq (1+\epsilon)f(x')$ (minimization) or $f(x) \geq (1-\epsilon)f(x')$ (maximization)–assuming non-negative objective values.

Notably, one may also want to track metrics that present combined information about solution quality and the time it takes to arrive at a particular quality. One example thereof is the *primal integral* [Ber13], which tracks the primal bound towards the optimal (or best known) solution throughout the iterations executed by a solver. This approach has been found to be suitable for comparing the performance of solvers that do not necessarily return optimal solutions. This includes machine learning-based optimization solvers [KvHW19, BMZ+24, BHP+25, LLL+24]–trained on a large number of known instances–which can, in certain cases, predict good solutions quickly.

3.3.2 Computational Resources

To compare different methods and algorithms, it is crucial to transparently report the type of computational resources used and the duration for which they were utilized. Thus, software versions and hardware specifications, such as the type of processing units used and the amount of memory available, must be reported, as discussed in more detail below.

First, the *total runtime (s)*, i.e., the total wall-clock time for running the considered algorithm using the reported hardware, excluding idling times, e.g., queuing for execution on a (quantum) computer while the remaining processing units are waiting. For stochastic algorithms that are repeated multiple times, the average total runtime over all repetitions should be reported. If the algorithm is parallelized and uses multiple processing units simultaneously, this should be noted in the hardware description but not translated into a single-core runtime. The definition of the total runtime, excluding idling times, can be easily measured and represents the runtime most relevant for practical applications. In certain cases, it may also be useful to report the time-to-solution (TTS), i.e., the time from the algorithm's start until the returned solution was found.

Second, to better understand how different hardware is utilized, reported runtimes should be split into the execution time on *CPUs*, *GPUs*, *quantum computers*, and *alternative hardware*, such as, for instance, Field Programmable Gate Arrays (FPGAs) or Ising machines. How to accurately define, measure, and report runtime on a quantum computer is discussed in Section 3.3.3 below.

There are many additional details of interest, such as the intermediate problem sizes after pre-solving or the time and resources spent in different stages of the workflow: pre-solving, pre-processing (circuit preparation, etc.), training of machine learning-based solvers, post-processing, etc. While it is difficult to capture all of these details in a submission table (see Table 10), we strongly encourage benchmark submitters to provide such details in an accompanying reference, such as a paper or repository.

3.3.3 Measuring Quantum Computing Runtimes

In the following, we define quantum computing runtime and how to measure it on the example of the IBM Quantum Platform [IBM, MCG24a]; see Figure 1. A quantum computer consists of multiple stages, including a runtime environment, such as the *Qiskit Runtime*, the control electronics, and the actual quantum chip. The payload sent to a quantum computer needs to pass these stages, and even though some steps are performed using classical computing, such as, e.g., compiling a quantum circuit to microwave pulses, we attribute them to the runtime on the quantum computer. Thus, we define the runtime of a *single run* of a quantum (sub-) routine as the total time it takes to receive the payload, prepare it, run the circuit on the QPU, and extract and return samples from the system by measuring the qubits. We would like to highlight that the number of samples drawn from a quantum circuit corresponds to a hyperparameter that has to be chosen as part of the considered quantum algorithm. Furthermore, we assume that circuits are provided using a topology and gate set that is natively supported by the quantum computer. Thus, while we include the time it takes to prepare such a circuit for execution, e.g., to microwave pulses, we explicitly exclude the time it takes to map a given circuit to the instruction set supported by a particular quantum computer. This additional transpilation time should be considered during classical pre-processing. Similarly, we attribute time spent on error mitigation to the quantum computer, if it happens as part of the runtime environment, and to the classical processing units, if it is handled via post-processing.

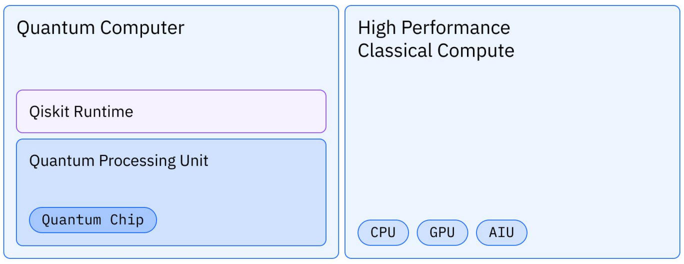


Figure 1: Illustration of a computational environment that provides access to quantum and classical resources. On the left side, it shows the components of an IBM Quantum computer: the Qiskit Runtime environment [JATK+24], the Quantum Processing Unit (QPU), and the actual quantum chip. The QPU consists of the quantum chip and control electronics such as interconnects, amplifiers, and signal filtering. The right side illustrates typical classical high-performance computational resources such as central processing units (CPU), graphical processing units (GPU), and artificial intelligence units (AIU), e.g., tensor processing units. These classical resources may be employed by an algorithm in synergy with the quantum resources to achieve the best possible result. Credits: IBM Quantum [GM24].

Last, we outline how to measure the quantum computing runtime in the most reliable way. When using an IBM Quantum computer, one should run quantum circuits using the Qiskit Runtime *Session*-mode [JATK+24] instead of *Batch*- or *Job*-mode. Once a session has been started, it blocks the whole quantum computer for a user, in contrast to batch and single-job mode, which may share resources–such as circuit preparation pipelines–across different users and, thus, may not provide accurate runtime estimates. After a job has been executed, the results returned from the IBM Quantum Platform [IBM] contain the session start and end times, as well as the times individual jobs were submitted, started, and completed. In addition, the results report actual usage, which refers to the time that was actually spent on the QPU, excluding classical pre- and post-processing inside the quantum computer. Due to the potential parallelization of circuit preparation pipelines before executing them on the QPU, we cannot just add the runtimes of the individual jobs. Instead, we take the total time the session is active, i.e., the difference between start and end time excluding initial queuing time, and further subtract all times where the session was active but idling, i.e., while no jobs were submitted and we were not waiting for previously submitted jobs to be completed. This is what we define as the quantum computer runtime, while the idling time in between jobs should be attributed to, e.g., the classical runtime.

4 Problem Classes & Instances

This section presents ten problem classes and corresponding instances that are non-trivial to solve with state-of-the-art methods–for most of them already at relatively small scales. Furthermore, the respective problem classes are mostly related to practically relevant problems. Such problems are hard to find for two reasons: First, unsolved problems of industrial relevance are rarely published. Second, problems of practical relevance that cannot be solved are usually discarded and replaced by artificial simplifications of the original problem. This leads to a *selection bias* and may be one reason why a benchmarking framework that includes truly difficult (for reasonable system sizes) and relevant (ideally in practical settings) optimization problems that enable a meaningful and fair comparison of quantum and classical algorithms has not been established until now.

Problem instances can be loosely classified into three categories, which are not always mutually exclusive: *Real-world instances* arise from real-world problems, and are therefore crucial for demonstrating a practical quantum advantage. Since there is already a demand for a solution, such instances have often been modeled (and possibly simplified) such that they can be tackled by existing solvers. The availability of real-world instances to the scientific community is often limited, even more so for unsolved problems as discussed above. *Random instances* are based on randomly generated data, which may be motivated by real-world problems or completely artificial.

Depending on the generation method, such instances often do not capture the structural peculiarities of a real-world setting, and may be easy or hard to solve. *Crafted instances* are most likely the best way to construct artificial examples that are hard to solve. However, there is a risk that the instances are of no practical relevance, as they often represent borderline pathological cases; see, e.g., [HZ21, VGMG23]. A quantum advantage for random or crafted instances is interesting, as it may help to identify real-world problems with similar structure. The problem instances presented here were mostly crafted to fit a range of criteria we identified as crucial for a useful optimization benchmarking framework, while they are still mostly related to real-world problems.

Problem instances can be further classified as *primal* difficult, *dual* difficult, or both. Primal difficulty refers to instances where it is difficult to *find* an optimal solution, dual difficulty refers to instances where it is difficult to *prove* a solution is optimal. An astonishing number of instances are primal easy but dual hard. Before solving an instance, it is a priori unclear which is the case. One may, however, gain intuition about the difficulty of a problem instance from solutions of other instances of the same problem class, since instances within the same class often behave similarly. Since most currently known quantum optimization algorithms are heuristics, we consider the chances for demonstrating a near-term quantum advantage higher for primal difficult problems and do not yet expect a benefit for proving optimality, although that might be possible later. Hence, we mainly present problems that are considered primal difficult. Further, if it is easy to find a very good or even optimal solution, the problem is not difficult from a practical point of view, and thus, there is less potential for demonstrating practical quantum advantage.

We applied the following criteria for the selection of the problems presented in the remainder of this section. First, we consider problems that can be modeled using only integer variables, preferably binary ones, and all coefficients can be represented as integers. In an all-integer setting, checking the feasibility of solutions is easy and does not suffer from issues with rounding or numerical accuracy, and integer variables are more amenable for known quantum optimization algorithms without requiring too many qubits. Second, problem classes should contain instances that are hard to solve for state-of-the-art classical approaches, preferably even at a relatively small size, and preferably because of primal difficulty. This enables testing known quantum optimization algorithms already today. Third, all selected instances have a feasible solution. For those problem classes that are not genuinely feasible, we generated the instances in a way that guarantees feasibility, as discussed in the following sections.

Table 2 provides an overview of the problem classes and a summary of their key properties. In particular, it highlights challenges that may be introduced when translating problem instances from Integer Programming to QUBO. This includes increasing the number of variables, conversion of sparse to dense problems, or significantly larger ranges of coefficients. This shows the importance of selecting the right modeling approach for a problem.

The set of problems was selected to be diverse and can be grouped into three categories:

- Market Split, Maximum Independent Set, and Network Design are classic binary optimization problems. They have been studied intensively before the year 2000, cf. corresponding sections. Since then, only limited efforts have been put into improving the state-of-the-art in targeted exact solvers or heuristics. We attribute this to the fact that general MIP solvers and known heuristics are good enough for many practically relevant settings. However, these problem classes are well-established, can be effectively solved at small sizes, and become difficult to solve exactly for growing dimensions. The latter is particularly true for Market Split and Network Design. It follows that these problems are great candidates for testing, tracking, and pushing the capabilities of quantum algorithms.

- LABS, Minimum Birkhoff Decomposition, Steiner Tree Packing, Sports Scheduling, and Topology Design are problem classes that received limited attention compared to, e.g., Traveling Sales Person, Independent Set, or Knapsack problems. Thus, it may be that the known state-of-the-art classical algorithms could still be significantly improved. At the same time, quantum algorithms may offer an interesting approach to solving these types of practically relevant problems, especially because LABS and Minimum Birkhoff Decomposition are already difficult at very small sizes. Steiner Tree Packing and Sports Scheduling become interesting at larger system sizes, but they have many practically relevant applications and, especially for the latter, even heuristics are struggling.

- Portfolio Optimization and Vehicle Routing problems have obvious practical applications. Companies face diverse variations of these problems depending on industry and operational execution. Hence, there are indefinitely many variations of these problems depending on individual constraints and/or objectives, resulting in problem instances of varying difficulty.

For this reason, it is difficult to define a small set of problem instances to benchmark and compare established approaches, which are representative of these problem classes. In addition, many established solvers from commercial software vendors are not accessible for open-source benchmarks. Despite the practical relevance of these problem classes, no well-established benchmark set exists for testing the progress of quantum algorithms. We aim to fill this gap by providing a selection of relevant problem instances. However, it should be noted that all given Vehicle Routing problem instances can be solved to optimality with existing methods. This is a shortcoming that we aim to address in the future.

The number of decision variables for which MIP or QUBO formulations of the selected instances become difficult ranges from less than 100 to 45,000, and is therefore within reach for testing algorithms on near-term quantum hardware. Depending on the problem, the ranges of the constraint coefficients lie between 1 and $\approx 10^7$, assuming that the coefficients are scaled to integers, as we do throughout the paper. This can pose another challenge for quantum computers as they need to be able to represent the problem data in the necessary resolution.

Figure 2 provides a more detailed overview of the instances provided for the various problem classes, complementing Table 2. For MIP and QUBO formulations of the considered problem instances, it illustrates the number of variables, the problem density, the coefficient range, and whether the optimal solution is known. Please note that if multiple MIP and QUBO models are available (see [QOBLIB]), we choose to represent the one resulting in the smallest number of variables. We would further like to point out that the QUBO illustration only shows those instances that could be converted from an MIP model into a QUBO model with fewer than $1.6 \cdot 10^5$ variables. As the plots show, the larger the number of variables, the more likely it is that we do not know the optimal solution. Finally, it can be seen that Market Split and LABS are the two problem classes for which difficult instances are found at the smallest sizes. Independent Set also has hard

	MIP							QUBO		
Problem	**Type**	**Dense**	**#Vars**	**Coeffs**	**Constr**	**Feas**	**Bin**	**Dense**	**#Vars**	**Coeffs**
Market Split	F / L	d	78	48	1	no	yes	d	70	$\sim 5 \cdot 10^6$
LABS	Q / Q	s	81	4	1	yes	yes	s	820	$\sim 4 \cdot 10^4$
Birkhoff	L / L	s	240	10^4	3	yes	no	d	3,480	$\sim 3 \cdot 10^{10}$
Steiner	L / L	s	423,360	3	9	no	yes	-	-	-
Sports	F / L	s	8,608	2	> 10	no	no	s	11, 791	$\sim 4.5 \cdot 10^3$
Portfolio	Q / L	s*	690	$\sim 3 \cdot 10^4$	2	yes	yes	d	690	$\sim 2 \cdot 10^9$
Independent Set	L / L	s	500	1	1	yes	yes	d	500	2
Network	L / L	s	1,211	10^6	5	yes	no	s	46,330	$\sim 2.5 \cdot 10^{19}$
Routing	L / L	s	-	-	< 10	yes	no	s	-	-
Topology	L / Q	s	2,176	2	$4 - 7$	yes	no	s	-	-

Table 2: Problem classes overview - details on illustrative MIP and QUBO formulations, as applicable. **Type** denotes the objective/constraint type as linear (L) or quadratic (Q) for either if the problem corresponds to an optimization problem. For feasibility problems, the objective type is indicated as feasibility (F). **Dense** denotes whether the model is dense (d), defined as $|A| \geq \frac{1}{4}nm$, where $A \in \mathbb{R}^{m \times n}$ is the constraint matrix of a MIP and n, m denote the number of variables and constraints, respectively, or sparse (s), otherwise. For QUBO models, density is defined as $|Q| > \frac{1}{8}n^2$, where we assume Q is an upper-triangular cost matrix (due to symmetry, and thus, the reduced pre-factor of $\frac{1}{8}$ instead of $\frac{1}{4}$ used for MIP). Again, n denotes the number of variables. Portfolio has a sparse constraints matrix but a dense quadratic objective, which we indicate by s^*. **#Vars** denotes the approximate number of decision variables of the models corresponding to instances where the problems can become difficult for classical methods to solve to proven optimality in MIP formulation. Hence, the number of decision variables in the QUBO column denotes the size of the QUBO formulation, where solvers struggle to solve the corresponding MIP. This is to highlight how certain problems become significantly larger when represented as QUBO instead of an MIP. **Coeffs** is the approximate coefficient range, i.e., the maximum range of all coefficients in the models. Again, we can have a significant discrepancy between MIP and QUBO representations for certain classes. **Constr** Number of different constraint types in the MIP formulation ($\neq$ total number of constraints). This indicates the complexity of modeling the problems. **Feas** denotes whether it is trivial to find or construct a (initial) feasible solution to the problem or not. For QUBO, all solutions are feasible, and thus, we drop the column. **Bin** indicates whether all variables in the MIP problem are naturally binary or not. This is always true for QUBO and, thus, not reported. If this is not the case, we see a discrepancy between the number of variables in the MIP and the QUBO models. For LABS, we have natural binary variables; however, we receive quartic terms in the objective from adding squared quadratic equality constraints as penalties that need to be decomposed into quadratic terms by adding binary variables. QUBO instances requiring more than $1.6 \cdot 10^5$ variables have not been generated due to computational restrictions. We indicate this in the respective columns (-). Further, no values are given for Routing problems since we were able to find the optimal solution for all tested instances.

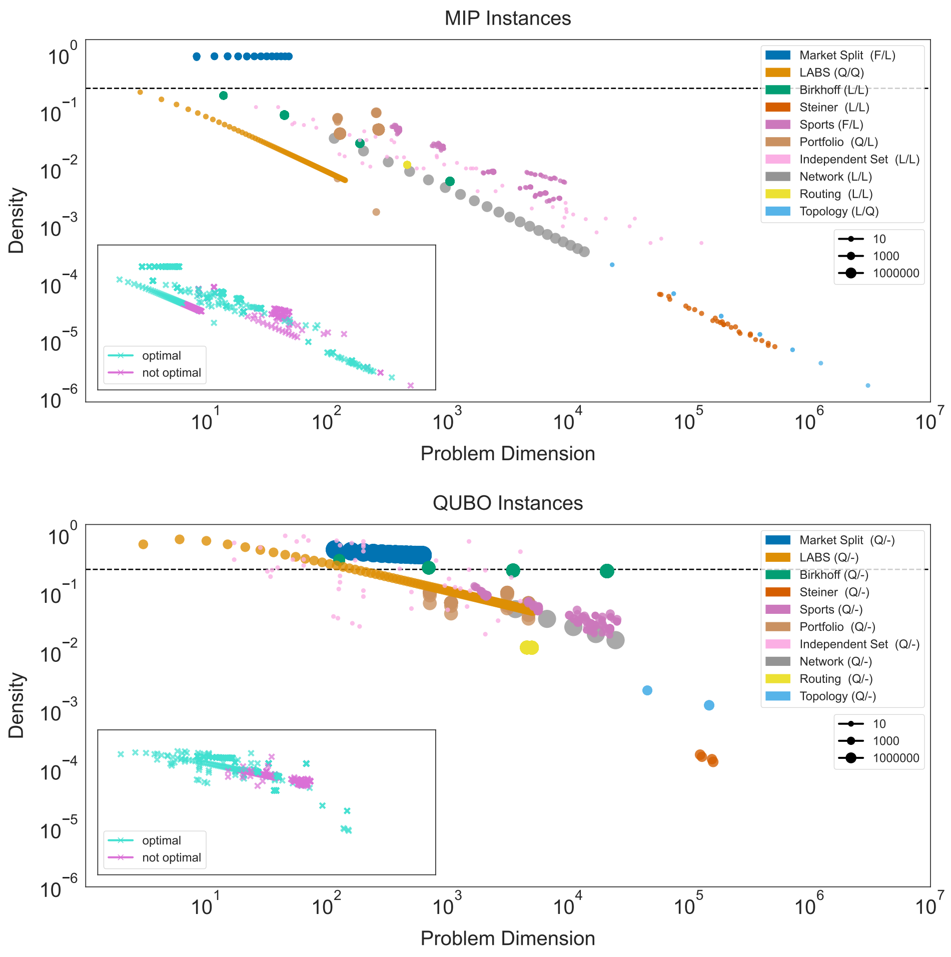


Figure 2: The figures plot the problem dimension (MIP: $\sqrt{\#variables \cdot \#constraints}$, QUBO: $\#variables$) against the density , i.e., the fraction of non-zero coefficients in the objective and, for the MIP instances, constraint functions, of the various problem instances provided in the [QOBLIB] repository for MIP (top) and QUBO (bottom) formulations of the problem. The legend entries for the various problem classes name the class itself and, in brackets, give the form of the (objective/constraint) function with L and Q denoting linear and quadratic, respectively. Furthermore, the size of the individual points is directly related to the coefficient range of the respective problem instance per problem class–see legend in the center right. The dashed lines illustrate the separation between sparse and dense instances as described in Table 2. Finally, the inset plots illustrate the instances in which the optimal solution is known. It should be noted that while the MIP plot shows all problem instances, the QUBO plot only shows those instances that we could transform into QUBO models with less than $1.6 \cdot 10^5$ variables, cf. Table 2. The respective model translation is done for Birkhoff, Steiner, Sports, Network, Routing, and Topology with Qiskit Optimization 0.6.1 [Qis] using the default parameters. Marketsplit, LABS, Portfolio, and Independent Set can be trivially converted to QUBOs using ZIMPL 3.6.1 [Koc04], since there are no slack variables to consider. Note that for Portfolio, this shows the density of the constraints matrix; the quadratic objective is always dense.

instances at relatively small sizes; these instances are also sparse and come with a small coefficient range. For these reasons, these three problem classes seem the most amenable for approaching with a quantum algorithm on near-term hardware.

In the remainder of this section, we discuss each problem class, describe the specific instances included, and present baseline results computed with classical optimization solvers. Of course, all instances, used models, and solutions are also made accessible in the [QOBLIB] repository. We do not claim that our baseline results are the best solutions one can find with a classical computer, but they should provide an indication of what is possible using general solvers without extensive effort. Moreover, the provided MIP and QUBO formulations were not subject to significant optimization efforts but are intended to serve as illustrations. Our goal is to provide a framework to track the progress towards quantum advantage in optimization and have an initial baseline to start with. To build-up expertise and start tracking today's capabilities, we describe an informed approach to construct warm-up examples from the provided problem instances in Appendix A.

4.1 Market Split

4.1.1 Background

The market split instances are multi-dimensional subset-sum problems, which were explicitly designed to be hard to solve using classical methods. Since their introduction by [GC99] in 1999, the progress in solving them has been moderate. Subset-sum is a classic **NP**-hard problem [GJ79]. The time complexity for enumeration is $\mathcal{O}(2^{n/2} \cdot (n/4))$ [SS81]. Several other methods have been tried, including smart enumeration, branch-and-cut, and lattice point enumeration with basis reduction [ABH+00, Was02, Vog12].

Why is it interesting?

Market split problems are a type of multi-dimensional subset-sum problem. This problem class contains various instances for which it is challenging to find feasible solutions with classical digital systems–already for small system sizes, with little more than 100 variables. Applications of the market split problem could be interesting in various industries, such as the energy sector, e.g., to help derive optimal pricing strategies subject to customer acquisition, retention or competition constraints, and regional regulatory policies [GMS+16, FHK+21].

4.1.2 Problem statement

Given a matrix $A \in \mathbb{N}^{m\times n}$ and a right-hand side $b \in \mathbb{N}^m$, for $m, n \in \mathbb{N}$, we want to find $x \in \{0,1\}^n$ such that

$$Ax = b. \tag{1}$$

This is a multi-dimensional subset-sum problem. We can transform Eq. (1) trivially into unrestricted optimization problems:

$$\min_{x\in\{0,1\}^n} (b - Ax)^2 \qquad \text{or} \qquad \min_{x\in\{0,1\}^n} \|b - Ax\|_\infty ,$$

where a feasible solution exists if the global minimum is zero. With careful choices of the coefficients, this problem becomes hard to solve, e.g., for numbers $m \in \mathbb{N}_+$, $D \in \mathbb{N}$, set $n = 10(m-1)$, $I = \{1, \dots m\}$, $J = \{1, \dots, n\}$ and $a_{ij} \in \{0, \dots, D-1\}, b_i = \left\lfloor \frac{1}{2} \sum_{j\in J} a_{ij} \right\rfloor$ [GC99].

The best results so far have been achieved using forms of lattice enumeration with basis reduction. [Was02, Was25] report solutions to selected instances up to $m = 14$. Traditional branch-and-cut-based MIP solvers often already struggle before size $m \geq 7$. However, they are typically more effective on the lattice reformulation introduced in [ABH+00].

4.1.3 Instances

For this benchmark, we generated four random instances each for $m = \{3, \dots, 15\}$, and $D = \{50, 100, 200\}$ as described above. While not all procedures to generate Market Split instances result in feasible problems (e.g., [GC99], see the study in [ABH+00]), we generated them such that feasibility is guaranteed. For $m \leq 7$, we randomly generate problem instances and only include them if a MIP solver can find a feasible solution. For $m > 7$, we start with randomly generated feasible solutions and then generate problem instances that have that point as a solution, while resembling the characteristics of the instances in [GC99]; details are given in Appendix C. In our tests, the resulting instances are still hard for classical solvers. Finally, we provide a routine in [QOBLIB] that, given an instance and a vector $x \in \{0,1\}^n$, checks the feasibility of the vector.

4.1.4 Classical Baseline

We provide modeling examples for an integer linear program and a QUBO, which can be run with any MIP or QUBO solver. For the ILP, we use the constrained formulation with positive slack, minimizing the slack as objective. For the QUBO, we add all constraints as quadratic penalties to the objective. Figure 21 in the Appendix depicts detailed runtimes by Gurobi 11.0.0 [gur25] and ABS2 [NTI+23] for instances with $3 \leq m \leq 7$.

Enumeration methods, either Lattice-based according to [Was02], or Schroeppel-Shamir [SS81], show better performance and allow solving all instances up to size $m = 11$ and partly up to $m = 14$. Figure 3 depicts a comparison of the runtimes of the algorithm from [Was25] on an Intel Xeon E-2288G CPU @ 3.70 GHz and the GPU-accelerated Schroeppel-Shamir algorithm [KK25] on an Nvidia GH200 system. Note that both methods are capable of computing all solutions to an instance. The time needed to find the "first" solution, however, varies easily by a factor of 10 between the instances.

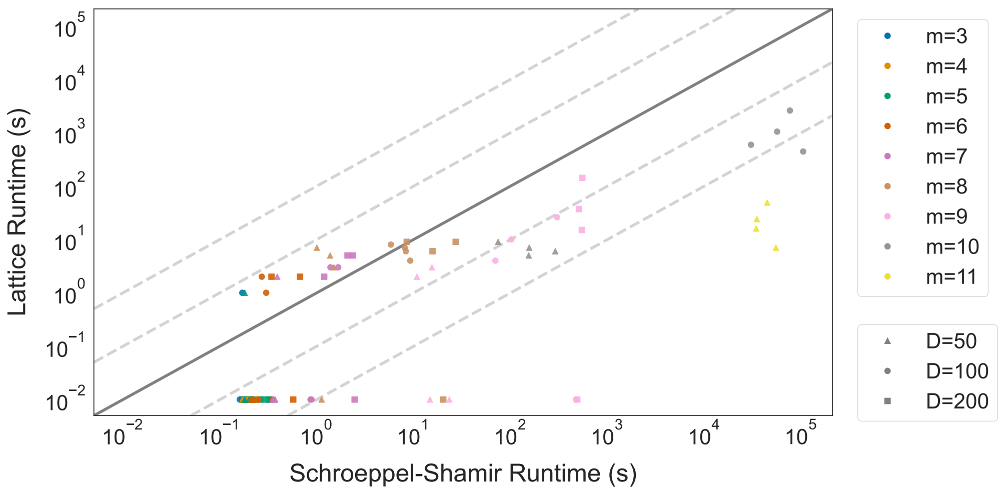


Figure 3: (Market Split) Comparison of runtimes using a lattice enumeration method and a GPU-accelerated Schroeppel-Shamir algorithm. The colors represent parameter m and the shapes represent parameter D.

4.2 Low Autocorrelation Binary Sequences

4.2.1 Background

Why is it interesting?

Established classical algorithms struggle to find (good) solutions to LABS instances even at relatively small sizes. Moreover, the binary (spin-like) structure of LABS aligns naturally with Ising-type models, making it a suitable test case for quantum optimization techniques. Demonstrating improvements in scalability or solution quality on LABS would represent a significant step in addressing a known hard problem.

Low Autocorrelation Binary Sequences (LABS) were initially studied in the context of radar and sonar system design, where achieving minimal autocorrelation sidelobes is crucial for improving range resolution and target discrimination [Boe67, Sch70]. Over time, these sequences have also proven useful in digital communications, coding theory, and spread-spectrum systems, where low autocorrelation properties can enhance channel efficiency and signal clarity. Unlike many hard optimization problems, each sequence length N in LABS defines a single, unique problem instance. As a result, progress on LABS is easy to measure: pushing beyond known solutions or improving run-time for larger N directly reflects meaningful advances in algorithmic capability.

LABS also appear as ground states of the Bernasconi model in statistical physics, a system involving complex, long-range four-body spin interactions [MB98]. Despite significant research efforts, finding optimal LABS remains a challenging task. LABS instances can be solved using convex-relaxation-based branch-and-bound, see, e.g., [ELL21]. In this line of research, LABS is formulated as a polynomial optimization problem over 0-1 variables and solved using branch-and-cut techniques. Both off-the-shelf solvers and specialized solvers can be used to deal with the corresponding formulation, but the exact solution of large instances remains difficult; see [ELL21] as well as the benchmark results (with optimality gaps) in [min24]. For convex-relaxation-based branch-and-bound, the empirical performance on LABS instances varies greatly, and it is difficult to estimate. The empirically-estimated running time of a tailored classical branch-and-bound method that uses combinatorial bounds [PM16] is approximately 1.73^N, while the best performing heuristic discussed in the literature for this problem (Memetic Tabu Search) has an estimated running time of approximately 1.34^N; see [SLC$^+$24] and the references therein for a discussion of these running times and [ZSKP25] for recent results. The work presented in [SLC$^+$24] also shows an empirical study, extrapolating from results on smaller instances, estimating the complexity of quantum optimization approaches, particularly those based on QAOA combined with amplitude amplification, to be around 1.21^N, considering idealized conditions. Although these figures are estimates for exponential-time algorithms, which are notoriously difficult to estimate correctly on unseen instances, they raise the possibility that quantum approaches could eventually outperform the fastest known classical algorithms.

4.2.2 Problem Statement

Given a sequence $S = (s_1, \dots, s_n)$ of length n with binary variables $s_j \in \{-1, +1\}$, the minimum energy autocorrelations of the sequence $j \in \{0, \dots, n-1\}$ corresponds to

$$\min_{s \in \{-1,+1\}^n} \sum_{j=1}^{n-1} \left(\sum_{i=1}^{n-j} s_i s_{i+j} \right)^2 .$$

As LABS is an unconstrained problem, any sequence S is a feasible solution.

4.2.3 Instances

Since there is exactly one instance per size n, all instances can easily be generated. We provide a checking routine that, given a sequence S, computes the autocorrelation energy and checks the result against known optima.

4.2.4 Classical Baseline

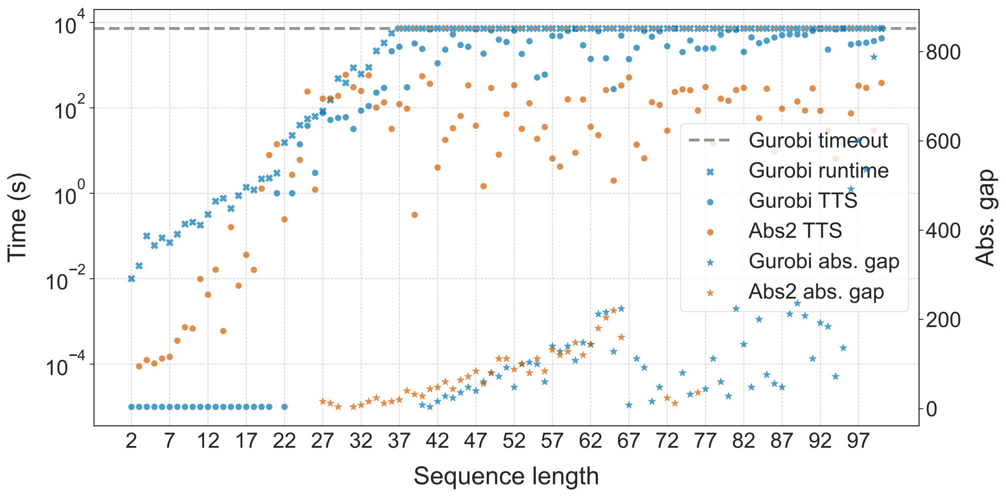


Figure 4: (LABS) Solution times using Gurobi and ABS2 for different LABS sequence lengths. *Abs. gap* denotes (the upper bound on) the absolute gap to the optimal (Gurobi)/best-known (ABS2) solution. Only gaps > 0 are shown.

Currently, the best-known exact solutions are limited to $n = 66$ and $n = 127$ for skew-symmetric variants; see [PM16]. We provide an example where LABS is modeled as a binary quadratically constrained quadratic program (BQCQP), which can be fed to a variety of established solvers. Figure 4 shows results obtained with Gurobi 11.0.0 on an AMD EPYC-7542 32-core processor using 64 threads and a time limit of 2 hours (we report the upper bound on the absolute optimality gap if optimality is not certified), and the results of ABS2 running on a system with two Nvidia A100 SXM4 80GB GPUs using a time limit of 10 minutes (we report the absolute gap to the best found solution). The largest instance that could be solved to optimality using this approach–although exceeding the usually set time limit–was $n = 41$. The respective execution time was 13 hours on a 72-core Nvidia GH200 system using Gurobi 11.0.2. An optimal solution was found in 6 hours and 47 minutes; the remaining time was used to prove optimality. Note that problem sizes up to $n = 19$ can be solved in less than one second and up to $n = 25$ in less than one minute. The optimal computed sequences are shown in Table 3.

Length	Sequence	Obj
2	2	1
3	12	1
4	112	2
5	113	2
6	1113	7
7	1123	3
8	1124	8
9	121113	12
10	111124	13
11	331211	5
12	111522	10
13	5221111	6
14	41112221	19
15	11213133	15
16	2112113131	24
17	11312144	32
18	1112115222	25
19	4111142212	29
20	11114142122	26
21	27221111121	26
22	11111212723	39
23	231131121413	47
24	122121111732	36
25	122121111733	36
26	3371111212211	45
27	34313131211211	37
28	112112131313431	50
29	112212111117323	62
30	132311111212164	59
31	1112111122122337	67
32	71112111133221221	64
33	742112111111122221	64
34	22229421121111111	65
35	11111132326612121	73
36	1121112121311132363	82
37	22228421121121111111	86
38	1222211111112112448	87
39	11111112343212112128	99
40	44412112131121313131	108
41	343111111222281211211	108

Table 3: (LABS) Optimal solutions obtained by Gurobi where each sequence encodes the counts of consecutive identical tokens in the corresponding solution. The counts are listed in order–reflecting the length of each uninterrupted group of the same token.

4.3 Minimum Birkhoff Decomposition

4.3.1 Background

The minimum Birkhoff decomposition problem is a special case of the Birkhoff decomposition problem [DU16]. In 1946, Birkhoff showed that any $n \times n$ doubly stochastic matrix can be written as a convex combination of permutation matrices [Bir46]. The method of proof used by Birkhoff inspired a class of algorithms for decomposing doubly stochastic matrices known as the *Birkhoff algorithm* [Bru82a, pp. 192], which can decompose a doubly stochastic matrix with at most $(n-1)^2+1$ permutation matrices [MR59]. It is possible to obtain tighter upper bounds on the number of permutations required depending on the number of zeros in the doubly stochastic matrix (see, e.g., [Bru82b]); however, the lower bound, i.e., the minimum number of permutation matrices required, is generally not known. In [DU16], Dufossé and Uçar showed that the problem of finding the minimum Birkhoff decomposition is **NP**-hard despite the existence of algorithms that can obtain approximate decomposition where the error decreases exponentially with the number of permutations in the decomposition [KLS17].

Why is it interesting?

The problem of finding the minimum Birkhoff decomposition is **NP**-hard [DU16], and it is hard to solve in practice, even for small instances. Also, it is easy to craft problem instances and obtain non-trivial upper bounds on the optimal cost, which is generally challenging in **NP**-hard problems. Another interesting aspect is that the problem of decomposing doubly stochastic matrices is well-studied, and many mathematical formulations and algorithms exist, including MIP and convex optimization. We also note that doubly stochastic matrices have a natural representation in terms of unitaries [MAT+24], which are the backbone of quantum computation.

Decomposing doubly stochastic matrices has applications in assignment problems, especially in networking [CCH00, YZLH17, LML+15, TZX+21, BVAV16]. In those problems, a permutation matrix corresponds to a matching in a bipartite graph, which represents a schedule or system configuration used to exchange some resource (e.g., data packets, energy). The weight associated with a permutation matrix captures the time the system will spend in such a configuration. Sparse decompositions are important to reduce the time a system spends switching between configurations, which is non-negligible in practice. Studying this problem can also be of interest for more general graph scheduling problems where the underlying graph is not just bipartite, e.g., in peer-to-peer networks [OFCCBM23].

The algorithms for solving the (minimum) Birkhoff decomposition methods can be divided into three classes. Branch-and-bound algorithms for solving a MINLP [HTBP22], Frank-Wolfe algorithms [CP23, Jag13], and variants of the Birkhoff algorithms [LML+15, VIT21]. Each approach has different characteristics. Branch-and-bound algorithms are often the slowest ones but can obtain certifiable optimal solutions when the doubly stochastic matrices are not too large or too dense. In contrast, FW algorithms are very fast and can obtain coarse approximations with just a few iterations. However, they are often unable to obtain exact decompositions even when the number of permutations used exceeds $(n-1)^2+1$. Birkhoff-type algorithms take an approach that can be considered between FW and branch-and-bound methods: they can obtain exact decompositions with at most $(n-1)^2+1$ permutations, and their speed is comparable to FW algorithms. Unlike branch-and-bound, Birkhoff-type algorithms do not guarantee that the solution obtained is optimal.

4.3.2 Problem statement

Let D be an $n \times n$ doubly stochastic matrix and P_i the i-th $n \times n$ permutation matrix, $i \in \{1, \dots, n!\}$. Recall that a matrix is doubly stochastic if its entries are non-negative and the rows and columns sum to one. Similarly, a permutation matrix is a doubly stochastic matrix with binary entries.

For a given doubly stochatic matrix D, the *minimum Birkhoff decomposition* problem is:

$$\begin{array}{ll} \min\limits_{c_i \in [0,s]} & \sum_{i=1}^{n!} |c_i|^0 \\ \text{subject to} & sD = \sum_{i=1}^{n!} c_i P_i \\ & \sum_{i=1}^{n!} c_i = s \end{array},$$

where we define $0^0 = 0$, i.e., $|c_i|^0$ counts the number of non-zero entries, and $s > 0$ is a scalar, often equal to one. The goal is to find a subset of permutation matrices such that D is in its convex hull and that this subset contains as few permutation matrices as possible. The following is an example of a decomposition of a 3×3 matrix:

$$\begin{aligned} \begin{bmatrix} 0.2 & 0.3 & 0.5 \\ 0.6 & 0.2 & 0.2 \\ 0.2 & 0.5 & 0.3 \end{bmatrix} = 0.2 \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} + 0.2 \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} + 0.1 \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \\ + 0.5 \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} + 0.0 \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} + 0.0 \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}. \end{aligned} \tag{2}$$

Note that $\sum_{i=1}^{n!} |c_i|^0 = 4$ in the example above since the decomposition only uses four out of the $3! = 6$ possible permutation matrices.

4.3.3 Instances

The instance set consists of doubly stochastic matrices of size $n \in \{3, \dots, 16\}$ generated by sampling n (sparse) and n^2 (dense) permutations uniformly at random. There are 10 instances for each matrix size and density. The weight associated with each permutation is also sampled uniformly at random with a fixed number of digits after the decimal points. The number of digits after the point depends on the size of the doubly stochastic matrix. We then scaled all coefficients to make them integers. See [QOBLIB] for a detailed description of how doubly stochastic matrices are generated. By construction, sparse doubly stochastic matrices can be decomposed with at most n permutations, which is a non-trivial upper bound on the minimal decomposition compared to the upper bound $(n-1)^2+1$ that applies to all doubly stochastic matrices. It should be noted that the instances were generated from feasible solutions to guarantee the feasibility of all given problem instances. However, it is not known whether these feasible solutions are optimal.

4.3.4 Classical Baseline

The goal of this section is to illustrate the performance and limitations of different approaches to set a performance baseline with state-of-the-art classical solvers for the minimum Birkhoff decomposition problem instances. We compare two convex optimization algorithms, Blended FW [BPTW19, Fra] and Birkhoff+ [VIT21, Bir], as well as CPLEX.

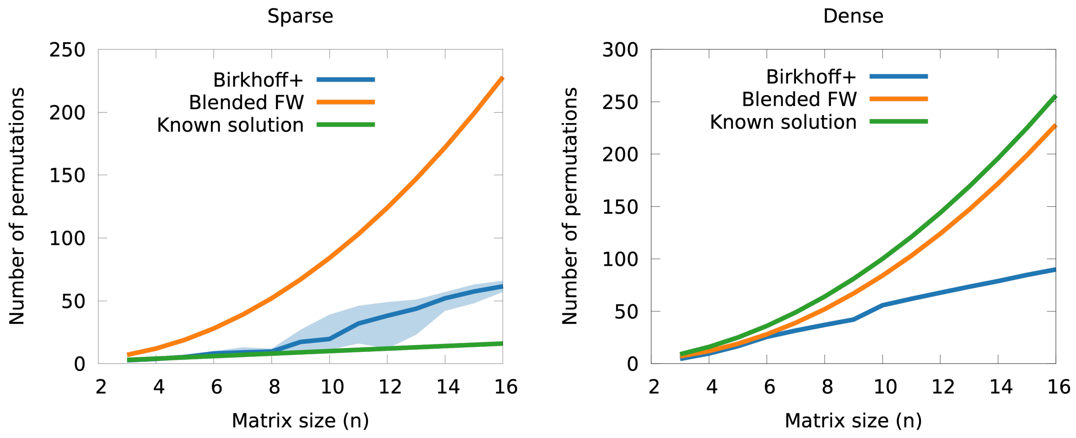


Figure 5: (Birkhoff) Illustration of the number of permutations required to decompose sparse (left) and dense (right) doubly stochastic matrices for varying matrix size. The shaded area–which is only visible in the sparse case–presents the maximum and minimum required number of permutation matrices per problem size.

First, we focus on the two convex optimization algorithms devised to obtain sparse approximate decompositions, i.e., not a minimal and exact decomposition. Notably, classical heuristics are discussed for this problem because we also investigate quantum heuristics in Section 5.2. We run the algorithms with their out-of-the-box settings [Fra, Bir] and show the results in Figure 5. The shaded area in Figure 5 (only visible in the sparse case) indicates the maximum and minimum number of permutation matrices required for a given problem size, i.e., the best and worst case instance. The 'known solution' corresponds to the number of permutation matrices used to generate the doubly stochastic matrix in the dataset, which is an upper bound to the minimum decomposition. Observe from the figure that Birkhoff+ obtains better performance than Blended FW in both cases. The reason for that is that Birhoff+ is guaranteed to obtain an exact decomposition with at most $(n-1)^2+1$ permutations, whereas Blended FW does not have such a property, even though it can reduce the approximation error exponentially fast with the number of permutations this adds to the decomposition. In the experiments, we limit the maximum number of permutations of Blended FW to $(n-1)^2+1$ and observe decomposition errors in between 10^{-2} and 10^{-6} depending on the instance. With Birkhoff+, all the decompositions are exact, and their performance varies depending on whether the matrix to decompose is sparse or dense. In the sparse case, Birkhoff+ obtains decompositions that match the number of permutations used to generate the matrices (i.e., n) when $n \leq 8$. For $n \geq 9$, the number of permutations required varies significantly depending on the instance (shaded area), especially in the range $n \in [9, 12]$. Finally, in the dense case, we obtain that the number of permutations obtained is smaller than the number of permutations used to generate the doubly stochastic matrix (i.e., n^2). There is no significant variability across instances, and the number of matrices required by Birkhoff+ to obtain an exact decomposition scales linearly up to the problem sizes investigated in this analysis.

Next, we illustrate the performance of CPLEX when solving the following Integer program (IP): $\min_{c_i,P_i} \sum_{i=1}^{k} |c_i|^0$ s.t. $sD = \sum_{i=1}^{k} c_i P_i$, $\sum_{i=1}^{k} c_i = s$, $c_i \in \{0, 1, \ldots, s\}$, $P_i \in \{0,1\}^{n \times n}$, $P_i \mathbf{1} = P_i^T \mathbf{1} = \mathbf{1}$ for all $i \in \{1, \ldots, k\}$ where $\mathbf{1}$ is the all-ones vector, and s and k integers. Note that the permutation matrices are formulated as decision variables, and all decision variables are represented by integers. Also, note that when the entries of D have a fixed number of decimal points (see Section 4.3.3), we can always select s to be an integer and make sD an integer-valued matrix. For example, if we multiply Eq. (2) by 10, we obtain $\left[\begin{smallmatrix} 2 & 3 & 5 \\ 6 & 2 & 2 \\ 2 & 5 & 3 \end{smallmatrix}\right]$, which we can decompose with weights 2, 2, 1, and 5. Parameter k in the formulation controls the decomposition length–which is ideally as small as possible. Thus, to solve the problem, we start the optimization with $k = 1$ and increase k by one if CPLEX fails to return a feasible solution. Figure 6 illustrates the runtime of CPLEX as a function of the matrix size. One can see from the figure that the runtime increases rapidly in both cases. With CPLEX on a laptop (M1 Max CPU and 32 GB memory) and a one-hour runtime limit, we can only solve instances up to $n = 4$ when the matrix is dense and up to $n = 8$ when the matrix is sparse.

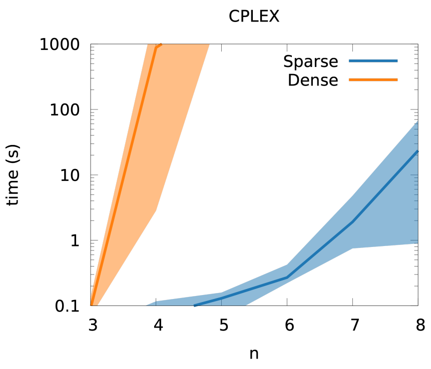


Figure 6: (Birkhoff) Arithmetic mean of time required by CPLEX to decompose a doubly stochastic matrix with the minimum number of permutation matrices. The x-axis indicates the size of the doubly stochastic matrix, and the shaded area indicates the minimum and maximum time required to decompose an instance of such size.

4.4 Steiner Tree Packing

4.4.1 Background

The Steiner Tree Packing Problem (STPP) describes the problem of placing n individual Steiner Trees node-disjointly into a graph. For $n = 1$ this is the Steiner tree problem, i.e., the combinatorial variant of the much older Euclidean Steiner problem. The Euclidean Steiner problem asks for the minimal tree that connects a given set of points in the plane: the special case with three points was discussed by Fermat before 1640, whereas the general version may have originated from Gauss; see [Aro96]. The Steiner tree problem replaces points with vertices in a graph with given edges and weights, and we are asked to find a minimum cost tree that connects a given set of terminals [Hak71, Lev71]. There is abundant literature on the Steiner Tree problem, which is already **NP**-hard: we refer to [Pol03, Lju21] for two excellent overviews, containing an extensive list of references and to [RK23] regarding the current state-of-the-art in solving them.

STPP provides a natural mathematical optimization formulation for Very Large Scale Integration (VLSI) design [GMW97, HKRV11]. Therefore, its efficient solution has tremendous practical interest. Early work on the STPP included polyhedral studies [GMW96a], separation routines for valid inequalities [GMW96b], and approximation algorithms [JMS03]. For a more recent discussion of computational results on difficult STPP instances, see [HK12].

Why is it interesting?

The minimization problem underlying the Steiner Tree Packing Problem comes with a linear objective function; the density is relatively sparse, the coefficients are all one, and the decision variables are binary. Computing optimal solutions, especially for dense routing areas, proved to be very hard, and little progress–apart from improvements in hardware and general solvers–has been achieved in the last two decades. This problem is also notoriously difficult for heuristics since, in a dense solution, one wrong decision can render the problem infeasible. In practice, this problem class can be used to model network design problems while considering cost minimization and reliability constraints.

4.4.2 Problem statement

For $n \in \mathbb{N}_+$, we write $[n] = \{1, \ldots, n\}$. In the STPP, we are given an undirected simple graph $G = (V, E)$, and $n \in \mathbb{N}_+$ disjunct terminal sets $T_1, \ldots, T_n$ with $T_i \subseteq V$ for all $i \in [n]$ and $T_i \cap T_j = \emptyset$ for $i, j \in [n]$, $i \neq j$. The goal is to find an edge set $S^* \subseteq E$ of minimal cardinality such that $(V(S^*), S^*)$ is a forest, where $\bigcup_{i\in[n]} T_i \subseteq V(S^*)$ and no vertex from T_i is connected to any vertex of T_j for all $i \neq j$. The general STPP with $n > 1$, as defined above, requires the inclusion of several disjoint Steiner trees into the same graph. A solution is a forest, and each terminal set T_i is connected by a single tree in the forest.

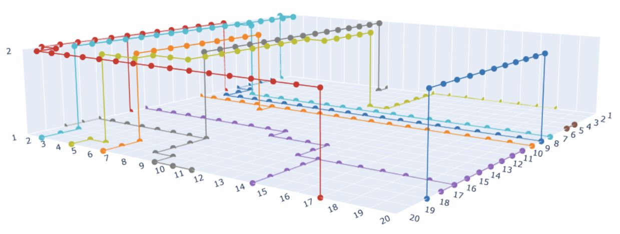


Figure 7: (Steiner) Representation of Steiner tree packing instance of size $20 \times 20 \times 2$ with the corresponding optimal solution, packing 8 Steiner trees, each one depicted in a different color. Here, for visualization purposes, we invert the z axis compared with the usual convention, i.e., we put terminals in the bottom layer.

4.4.3 Instances

We generated 3-dimensional $S \times S \times L$ grid graphs with holes. All terminals are placed at the border of the top layer. We require all trees to be node-disjoint. The parameters for each problem are the size of the grid graph S, the number of layers L, the maximum number of terminals per terminal set T, and the number of holes in the grid H. The holes can extend to multiple layers. An example of a $20 \times 20 \times 2$ instance with the corresponding optimal solution is given in Figure 7. In practice, the problem can also be defined with edge weights. For example, changing from one layer to another could be more expensive. However, since this does not make the problem more difficult in general, we kept it simple and set all edge weights to one. To ensure feasibility, the instances were reverse-engineered from feasible solutions, which may not be optimal. We observed that Gurobi started having difficulties with instances of size larger than $S = 20$.

Finally, we provide a program that checks if a given solution to the STPP is feasible for a given instance, i.e., that each network's terminals are connected by a Steiner Tree that is node-disjoint from all other Steiner Trees.

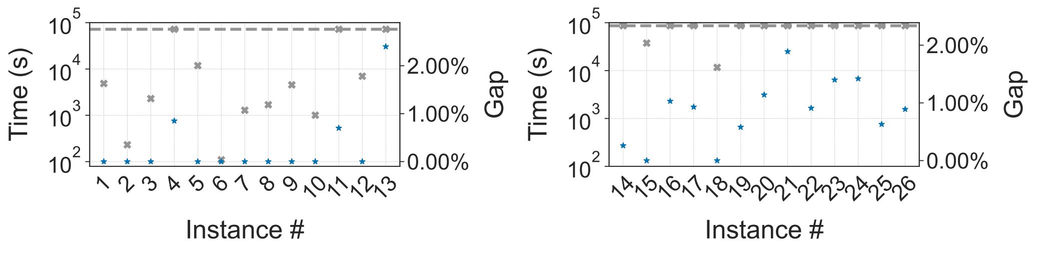


Figure 8: (Steiner) Total runtime and gap to the best-known (or even provably optimal) solution, solved with Gurobi for different size 20 instances with a timeout of 72000s (left) and different size 30 instances with a timeout of 86400s (right). The ticked line represents the timeout, the crosses indicate the runtime, and the stars show the gap.

#	Instance (size 20)	Sol.	Bd.	Gap	Time	#	Instance (size 30)	Sol.	Bd.	Gap	Time
1	stp_s020_l2_t3_h2_rs24098	228	228	0.00	4852.66	14	stp_s030_l2_t3_h1_rs97531	389	388	0.26	–
2	stp_s020_l2_t3_h3_sd97531	219	219	0.00	231.00	15	stp_s030_l2_t3_h3_rs97531	388	388	0.00	37495.02
3	stp_s020_l2_t4_h0_rs24098	242	242	0.00	2292.39	16	stp_s030_l2_t4_h2_rs97531	387	383	1.03	–
4	stp_s020_l3_t3_h2_rs24098	355	352	0.85	–	17	stp_s030_l2_t5_h1_rs24098	430	426	0.93	–
5	stp_s020_l3_t4_h0_rs24098	230	230	0.00	47150.22	18	stp_s030_l3_t3_h1_rs97531	327	327	0.00	11652.86
6	stp_s020_l3_t4_h2_rs97531	163	163	0.00	109.33	19	stp_s030_l3_t3_h2_rs97531	343	341	0.58	–
7	stp_s020_l3_t4_h3_rs97531	217	217	0.00	1286.76	20	stp_s030_l3_t4_h0_rs97531	439	434	1.14	–
8	stp_s020_l4_t3_h3_rs97531	268	268	0.00	1692.55	21	stp_s030_l3_t5_h1_rs24098	476	467	1.89	–
9	stp_s020_l4_t4_h0_rs24098	281	281	0.00	4543.95	22	stp_s030_l4_t3_h1_rs97531	328	325	0.91	–
10	stp_s020_l4_t4_h3_rs37235	189	189	0.00	1005.19	23	stp_s030_l4_t4_h0_rs97531	429	423	1.40	–
11	stp_s020_l5_t3_h3_rs24098	284	282	0.70	–	24	stp_s030_l4_t4_h2_rs97531	424	418	1.42	–
12	stp_s020_l5_t4_h0_rs24098	281	281	0.00	6973.54	25	stp_s030_l5_t3_h1_rs97531	320	318	0.63	–
13	stp_s020_l5_t4_h3_rs24098	333	325	2.40	–	26	stp_s030_l5_t4_h0_rs24098	447	443	0.89	–

Table 4: (Steiner) Results for instances of size 20 (left) and 30 (right). Sol. is the best solution objective found, and Bd. is the best lower bound found. Gap is the gap to the optimal solution, and Time is the total runtime in seconds, where – indicates a timeout (20 hours for size 20, 24 hours for size 30).

4.4.4 Classical Baseline

Using the multicommodity flow ILP formulation for the node-disjoint STPP given in [HK12], we ran Gurobi 11.0.2 on an AMD EPYC-7513 32-core processor with 64 threads to attempt to solve the smallest instances in our benchmark set (gridsize 20 or 30). We used the model "as is", without additional valid inequalities or decomposition strategies (the model is also provided in the repository [QOBLIB]). Hardware information and results are summarized in Figure 8, whereas solution values and bounds are given in Table 4. With this ILP model, we could solve most of the size 20 instances (with an average solution time of 53 minutes) within 20 hours, but only two of the size 30 instances within 24 hours (the solution times were 194 minutes and 624 minutes). Reimplementing the techniques of [HK12] using current solver technology, one may be able to solve a few more instances, but the largest grid sizes in the benchmark set remain out of reach for current exact methods.

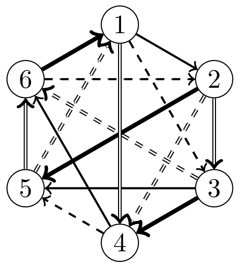
4.5 Sports Tournament Scheduling

4.5.1 Background

Why is it interesting?

As soon as any side constraints are considered, the complex combinatorial structure makes it very difficult for established heuristic methods to find feasible solutions. For several medium-sized instances, no known (and investigated) method was able to find any feasible solution [VG23, BGC$^+$23]–despite the fact that the generation process ensures the existence of feasible solutions. Furthermore, the problem is well aligned with real-world applications, which typically 'only' differ in terms of a few additional side constraints.

The design of algorithms to construct sports timetables goes back to at least the 1950s (see, e.g., [Fre56]). Particularly well researched are so-called single round-robin sports timetables where each team plays every other team once. Even though a single round-robin timetable can be constructed in time polynomial in the number of teams (see, e.g., [dW81]), the problem becomes **NP**-hard as soon as some elementary constraints are added (see, e.g., [BDS10]). One of the prime difficulties in sports timetabling is the vast size of the solution space: every single round-robin timetable corresponds to an (oriented) 1-factorization of K_n and the other way around (see Fig. 9), where K_n denotes the complete graph on n nodes. Considering just 14 teams and even ignoring the order of rounds, the permutation of team names, and the home-away status of games, more than 1,132,835,421,602,062,347 unique, corresponding timetables already exist [KÖ09].



S1	S2	S3	S4	S5
1-2	1-3	1-4	5-1	6-1
3-5	4-5	2-3	2-4	2-5
4-6	6-2	5-6	3-6	3-4

Figure 9: Oriented one-factorization of K_6 (left) and its associated single round-robin timetable (right), where each $i - j$ represents a home game of team i against team j for time slots 1 to 5.

Numerous IP-based methods have previously been proposed to construct sports timetables. See, e.g., [vHLS23] for a discussion on IP formulations, [NT98] for a popular decomposition approach known as first-break-then-schedule, and [VBG23a] for an application of Benders' decomposition.

At the same time, several QUBO-inspired algorithms have been proposed for a related sports tournament problem known as the break-minimization problem. In this problem, the objective is to minimize the overall number of breaks (i.e., consecutive games at home or away), given that the pairing of opponents is fixed in each round (e.g., [FM23, KKN22, UYS18]).

4.5.2 Problem statement

Given a set of teams $T = \{1, \ldots, n\}$ and time slots $S = \{1, \ldots, 2(n-1)\}$, n even, a double round robin (2RR) corresponds to a perfect matching of teams in each time slot such that over all time slots each team plays precisely once at home against every other team. We call a 2RR 'phased' if each team meets every other team exactly once during the first and last $n-1$ time slots.

The feasibility version of the ITC2021 sports scheduling problem considered in this benchmark is to find a phased 2RR that additionally respects the following hard constraints:

- **Capacity constraints.** Regulate when teams can play home or away.
  - CA1: team $i \in T$ plays at least or no more than k home games in time slots $P \subseteq S$;
  - CA2: same as CA1 but considering opponents as well;
  - CA3: no more than two consecutive home or two consecutive away games;
  - CA4: teams in $T_1 \subseteq T$ play no more than k home games against teams in $T_2 \subseteq T$ during time slots in $P \subseteq S$.
- **Break constraints.** A team has a break if it plays consecutively at home or away.
  - BR1: team $i \in T$ has no more than k breaks during time slots in $P \subseteq S$;
  - BR2: no more than k breaks in total.
- **Game constraints.** Forbid some games during specific time slots.
  - GA1: at most k games from $G \subseteq T \times T$ during time slots in $P \subseteq S$.

The original version of the ITC2021 sports scheduling problem also allowed the constraints above to be soft, penalizing violations of the soft constraints while strictly adhering to the hard ones. Additionally, the original version of the problem considered two more soft constraints:

- **Fairness and separation constraints.** Increase the attractiveness and fairness of the tournament.
  - FA2: at any point in time, the difference in the number of home games played between any two teams does not exceed two.
  - SE1: there are at least 10 time slots between each pair of games involving the same teams.

4.5.3 Instances

The problem instances stem from the International Timetabling Competition 2021 [VBG23b] and a follow-up study that generated additional problem instances using the same format [VGC+24]. These problem instances were generated using 'Instance Space Analysis' [SMMn23]. They feature a realistic set of constraints for real-world applications. Furthermore, they are challenging for state-of-the-art MIP and even dedicated sports tournament solvers. Finally, they feature a wide variety of characteristics that can expose different strengths and weaknesses of algorithms; see, e.g., [VGC+24]. Moreover, the instances were constructed so that a feasible solution is known to exist. More specifically, we made use of the fact that generating a basic sports schedule is easy, even when some of the constraints are considered. Suppose a problem instance should suffice p 'easy' constraints from type A and q 'difficult' constraints from type B. Then, we started by generating a solution that satisfies constraints A and added constraints of type B to this solution–such that the respective solution is compatible with constraints of type B.

The problem instances in this benchmark can be divided into four subclasses: ITC2021, Large, Medium, and Small. The ITC2021 class corresponds to the 45 original problem instances from the International Timetabling Competition on Sports Timetabling 2021 [ITC], considering all hard and soft constraints. Within the ITC2021 class are multiple instances for which no feasible solution was reported within a reasonable amount of time. The Large, Medium, and Small classes represent feasibility versions of the ITC2021 problem instances. More specifically, to lower the entrance barrier and simplify the problem, we ignore all soft constraints and, therefore, reduce the optimization problem to its feasibility version. For further experimentation, we also generated an additional set of problem instances having exactly the same number of constraints as the original ones; however, with 6, 8, and 12 teams only. These instances are contained in Medium and Small.

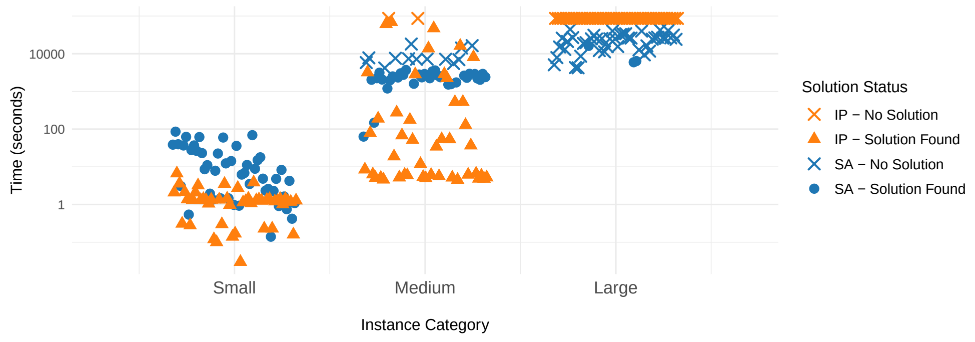


Figure 10: (Sports Scheduling) Solution times with Simulated Annealing (SA) and CPLEX (ILP) for the feasibility version of the problem. The time limit for SA was iteration-based, while the ILP solver timed out after 24 hours.

4.5.4 Classical Baseline

We provide feasible solutions to all problem instances–which were obtained during the instance generation. An online solution checker is available in the validator section of [ITC]. Furthermore, the repository also includes a MIP formulation modeling example proposed in [BKS21].

Figure 10 shows the average solution times for a state-of-the-art Simulated Annealing (SA) algorithm (see [RPDGS22]) and for a MIP formulation similar to the one presented in [BKS21]. Both algorithms were run on a machine equipped with an AMD EPYC-7532 32-core processor, using 8 cores for solving the MIP (CPLEX 12.10) and a single core for the SA algorithm. The MIP solver was given a time limit of 24 hours, while the SA algorithm was stopped after a fixed number of iterations. The MIP performed reasonably well on small and most medium-sized instances but failed to solve any of the large instances. Also, the SA algorithm managed to find only solutions to three large instances, even though it was given considerable computational resources. As a reminder, the instances were selected in such a way that it is known that existing state-of-the-art solvers heavily struggle to find feasible solutions.

4.6 Portfolio Optimization

4.6.1 Background

The problem of quantitative investment and risk diversification in portfolio optimization spans a broad spectrum of methodologies, each incorporating different simplifications, investor priorities, and trade-offs between risk exposure, returns, computational complexity, robustness, and out-of-sample performance. A foundational approach is Markowitz's mean-variance optimization, formulated as a convex quadratic program [Mar52]. This framework has been extended to address real-world constraints [KTF14], estimation errors [Haf80, CR83, LCBP99], sensitivity to input variability [MM08, Pra19, BTN99, TK04], and Bayesian techniques that integrate expert judgment with statistical models [BL92, AZ10, CPT18]. Computational limitations of the methodologies for practically relevant cases [MM15] have led to the adoption of, e.g., fixed-mix strategies that improve model transparency and stability [Mer69, DGU07], hierarchical risk parity and risk budgeting techniques [LdP16, GT22], and multiperiod approaches which help to capture intertemporal effects and hedging demands [RA00, GP13]. In fact, portfolio optimization models can be defined with varying complexity [PPSHW15, KR19, LUM22, HP23] depending on whether one aims to also jointly model risk, return predictability, and impact costs over time while addressing the exponential growth of time complexity with respect to the number of assets and rebalancing periods [BS11]. Portfolio optimization can transform into a stochastic dynamic programming problem, which, under realistic conditions that account for, e.g., transaction costs or short selling, has been proven to be computationally intractable for portfolios with more than two risky assets [Con79, BSS10].

Given its relevance and complexity, portfolio optimization has attracted considerable interest from the quantum computing community. Theoretical research suggests that quantum algorithms may offer speedups under specific conditions that may, however, not hold in real-world instances; see, e.g., [KPS19, RHG$^+$16, LR24]. Another line of research has investigated how one may approach simple portfolio optimization instances with near-term compatible quantum algorithms [BBD$^+$22, SMMW21, MKS$^+$22]. The simplified version of the problem derived here focuses on the computational complexity in the multiperiod setting, using a binary quadratic formulation of the mean-variance model with linear constraints, and perfect knowledge of expected future returns and covariances. This allows for the creation of compact yet computationally challenging problem instances, as the problem is **NP**-hard [Bie96], and the systematic benchmarking of current quantum algorithmic capabilities.

Why is it interesting?

A vast array of industrial and scientific domains are interested in solving problems where one aims to find an optimal composition of a portfolio of coupled objects with respective observables that typically share a common probability space. (Financial) Portfolio management is one of the hardest variants of this problem. It concerns the risk-sensitive allocation of a time-dependent investment budget for a number of diversified financial assets whose behavior varies with time according to unknown stochastic processes, which are driven by external factors. The hardness of this problem can be explored at different levels of complexity, even at a small scale, and, as such, allows systematic tracking of algorithmic and hardware progress.

4.6.2 Problem statement

Let $\mathcal{T} := \{t_1, \dots, t_m\}$ be an ordered set of discrete time points such that $0 \le t_1 < t_2 < \dots < t_m$, representing a finite investment horizon in the future. Consider an artificial deterministic market that is fully described by a closed system of $n \in \mathbb{Z}^+$ financial assets $\{a_i \mid i = 1 \dots, n\}$, where each asset a_i has known expected market prices $\{\bar{p}_{it}\}$ at each time $t \in \mathcal{T}$. Assuming a myopic investor in a mean-variance framework and no access to additional information, the objective is to find a multiperiod trading strategy $\mathcal{S}_1 := \{x_{it}\}$ at t_1, where portfolio allocations $x_{it} \in \{0, 1\}$ represent binary investment decisions of up to C normalized capital units $u \in \mathbb{R}^+$ into assets $\{a_i\}$. We define the price p_{it} as follows: Let $e_i := u/\bar{p}_{it_0}$, then $p_{it} = e_i\bar{p}_{it}$, i.e., we compute how many assets of type i we can buy for one unit of cash at time t_1. The amount of stocks e_i we can get at time t_1 for one unit of cash u is then used for all later time steps, i.e., p_{it} tells us what has become out of one cash unit invested into asset i at time t. The investor seeks to balance returns and risk at terminal time t_m, where the temporal returns are given by $(p_{it} - p_{i,t-1})x_{it}$ and the temporal risk is measured through the risk preference λ-weighted positive semidefinite covariance matrix Σ as $\lambda(px)^T\Sigma xp$, while obeying a set of regularizing trading constraints.

These constraints aim to incorporate more realistic real-world conditions, although here simplified into a quadratic optimization problem:

- Transaction cost: Rebalancing of x over $\mathcal{T}$ incurs costs due to fees, taxation, etc. that negatively offset the returns by a factor of transaction cost $\delta \in \mathbb{R}^+$ for each change $p_{it}|x_{it-1} - x_{it}| = p_{it}(x_{it-1} + x_{it} - 2x_{it-1}x_{it})$ and a liquidation cost $\delta p_i x_i$ at terminal time t_m. This is a strong simplification, as cost factors are neither global nor time-invariant scalars, but rather stochastic functions driven by market uncertainties.

- Short selling cost: Given a subset $f \in \mathcal{F} \subseteq \{a_i\}$ of assets permissible for short selling positions over $\mathcal{T}$, a fixed loan rate for short positions $\rho \in \mathbb{R}^+$ is applied to each allocation $\rho p_{ft} x_{ft}$ with $f \in \mathcal{F}$.

- Cash flow limits: Let $u \in \mathbb{R}^+$ denote the unit value of cash, and C the total available units of normalized capital. Each asset is assigned an indicator $\tau_i \in \{-1, +1\}$, representing a short (-1) or long $(+1)$ position. At any time, the total net investment $\sum_i \tau_i x_{it}$ must not exceed the capital limit C, and the total number of assets held at each time step must be bounded by B, which serves as an upper limit on the number of acquired positions.

- Cash interest: Temporarily unused capital earns risk-free interest $\nu u(C - \sum_i \tau_i x_{it})$ with a risk-free rate $\nu \in \mathbb{R}^+$.

- Multiple shares: We also allow that each asset i can be selected up to k times. We use a unary encoding where the Hamming weight of k binary variables encodes the number of shares, i.e., we essentially consider k copies of the same asset that can be selected. For the small number of shares considered here, this is as efficient as other encodings, but it has the advantage that we can easily model the transaction costs. Thus, we expand the vector of portfolio allocations to a $2kn$-dimensional binary vector $\mathbf{x}$ for the n assets at each time t, where each block of k binary variables represents a single asset type. The inclusion of a factor of 2 accounts for the allowance of short selling, with an equal number of vector elements allocated to represent both long and short positions.

The problem can then be formulated as a Binary Quadratic Program (BQP), where the objective is a quadratic function subject to linear constraints, and the decision variables are restricted to be binary:

$$\begin{aligned}\min_{x\in\{0,1\}^{n\times m},\, y\in\{0,1\}^{|\mathcal{C}|\times m}} \quad & \sum_{t=1}^{m}\Bigg(\lambda \overbrace{\sum_{i=1}^{n}\sum_{j=1}^{n}\tau_i p_{it} x_{it}\sigma_{ijt}\tau_j p_{jt} x_{jt}}^{\text{risk}} \\ & - \sum_{i=1}^{n-1}\underbrace{\tau_i(p_{i,t+1}-p_{it})x_{it}}_{\text{return}} - \sum_{i=2}^{n}\underbrace{\delta p_{it}(x_{it-1}+x_{it}-2x_{it-1}x_{it})}_{\text{transaction cost}} \\ & \underbrace{-\,\nu u\sum_{c\in\mathcal{C}}2^c y_{ct}}_{\text{cash interest}} + \underbrace{\rho\sum_{f\in\mathcal{F}}p_{ft}x_{ft}}_{\text{short selling cost}}\Bigg) + \underbrace{\delta\sum_{i=1}^{n}(p_{i1}x_{i1}+p_{im}x_{im})}_{t_1 \text{ transaction + liquidation cost}}\end{aligned}$$

subject to

$$\sum_{i=1}^{n}\tau_i x_{it} + \sum_{c\in\mathcal{C}}2^c y_{ct} = C \qquad \text{for all } t\in\mathcal{T} \tag{3}$$

$$\sum_{i=1}^{n} x_{it} \le B \qquad \text{for all } t\in\mathcal{T} \tag{4}$$

where (3) describes the capital limit and (4) limits the number of assets that can be included in the portfolio. Since we restricted the problem to binary variables, we modeled the amount of available cash units as binary variables $y_{ct}\in\{0,1\}$ with a suitable $c\in\mathcal{C}:=\{0,\dots,\lfloor\log_2(C)\rfloor\}$ such that $\sum_{c\in\mathcal{C}}2^c \ge C$. It is worth noting that alternative formulations may also be possible.

4.6.3 Instances

	Parameter	Value
ν	risk-free rate	0.01%
δ	transaction cost rate	0.1%
ρ	loan rate on short positions	0.0025%
C	available # of cash units	10 units
k	Max. # units of each asset	3

# inst	n	m	u	B	# vars
4	10	10	3	4	710
4	10	15	3	4	1065
4	50	10	3	20	3110
4	50	15	3	20	4665
4	200	10	3	50	12110
4	200	15	3	50	18165
4	400	10	3	100	24110
4	400	15	3	100	36165

Table 5: (Portfolio) (left) Fixed parameters used to generate the portfolio optimization problem instances. The annual rates of ν and ρ correspond to 2.55% and 0.92%, respectively. The total amount of available cash equals \$1,000,000. (right) Overview of the different sizes and parameters of the Portfolio Optimization instances for binary variables. We generated four instances for each setting.

Table 5 (left) outlines the parameters that are used in our experiments. The parameter choices are based on established literature [GKVdW05, D'a02, Tep00]. Using these parameters, we generated 4 instances each (1 containing the original data and 3 with randomly perturbed data) for 10, 50, 200, and 400 possible assets with two different time horizons: 10 days and 15 days. We used real S&P 500 data from January 1, 2024, to May 31, 2024, for the asset prices p_{it} and the corresponding covariances σ_{ijt}. The size of the instances is defined by the number of assets n, the number of considered time steps m, and the number of chosen asset units k. The risk-affinity/aversion parameter λ, the unit limit on asset selection B, and the capital bound C have a strong influence on the solvability of the instances. We would like to highlight that the repository includes instances with $\lambda = 0$, which represents an edge-case setting that lacks operational meaning but serves as a simplified mathematical representation.

Moreover, we generated randomly perturbed instances based on real-world S&P 500 data from January 1, 2024, to May 31, 2024. The generated data is sampled independently at each time step using the t-distribution with $\nu = 5$, scaled to the mean and covariance of the stocks as given by the original data, with a rolling window of 30 days. Outliers are randomly added to the dataset.

We always consider the n largest companies, ranked by their market capitalization as of January 1, 2025. Table 5(right) gives an overview of the instances in the problem class, with the instances on top being randomly generated and the instances on the bottom representing real-world data.

4.6.4 Classical Baseline

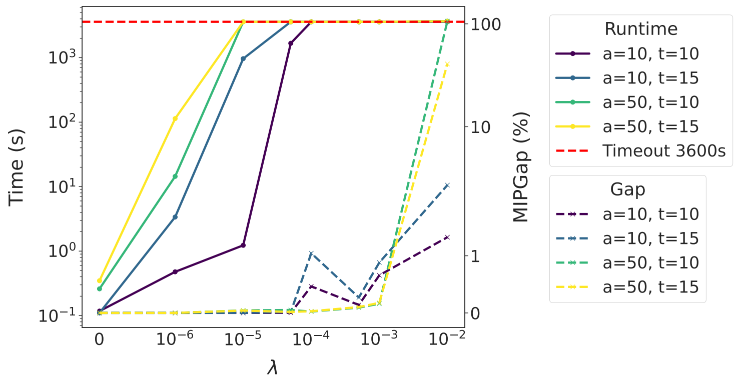


Figure 11: (Portfolio) Runtimes (solid lines) and MIP Gap (dashed lines) for Gurobi with a time limit of 3600s for different values of λ averaged over three randomly generated similar instances.

	Gurobi				Abs2		
λ	LB	Incumbent	Gap (%)	RT (s)	Incumbent	TTS (s)	Gap (%)
0	-879572	-879572	0.00	0.4	-879572	90.0	0.00
0.000001	-872141	-872055	0.01	398.7	-872055	79.1	0.00
0.00001	-887265.36	-819899	8.21	3600.1	-819899	30.2	0.00
0.00005	-692341.57	-680124	1.80	3600.2	-680124	50.7	0.00
0.0001	-604495.30	-586823	3.01	3600.2	-586823	206.7	0.00
0.0005	-424863.29	-386990	9.80	3600.1	-384862	1497.6	0.55
0.001	-367893.74	-314736	16.90	3600.2	-299298	741.9	4.91
0.01	-141194.79	-437920	3124.21	3661.8	-2825	1510.8	99.35

Table 6: (Portfolio) Comparison of (classical) BQP using Gurobi 12.0.1 with a tolerance of 1e-4 vs.(quantum-ready) QUBO formulation of a portfolio optimization problem–a050_t15_s00 instance. The ABS2 Gap is the relative gap between the ABS2 incumbent and the gurobi incumbent.

The tests are conducted by running the BQP model with Gurobi 12.0.1 on an Intel Xeon Gold 6338 CPU @ 2.00GHz processor using 128 threads with a timeout of 3600s and using the QUBO model with ABS2 on a system with four A40 SXM4 48 GB GPUs with a timeout of 3600s. An overview of different parameters up to $n = 50$ assets can be seen in Figure 11. With increasing λ, the running time and the upper bound on the optimality gap (MIP Gap) also increase. Table 6 shows execution details on the instance for $n = 50$, $m = 15$, $k = 3$, $B = 20$, $C = 10$: All the objective function values shown are negative because we are minimizing the risk, and therefore, the profit is subtracted in the objective function. As may be expected, when the risk adversity increases, the profit decreases, as well as the gap between the lower and upper bounds of the solution. The computational performance varies depending on the risk aversion parameter λ. For $\lambda = 0$, the risk term is ignored, and only the profit is optimized. This can be quickly solved to optimality. In the risk-averse case with a high $\lambda = 0.01$, the opposite happens. The lowest risk strategy is to take the risk-free interest rate $\nu = 0.01$ and receive a profit of $C \cdot m \cdot \nu = 1000$.

Figure 12 shows an in-depth analysis for a problem instance with $n = 50$, $m = 15$, $k = 3$, $B = 20$, $C = 10$. More specifically, the Figure shows how the objective value changes over time for $\lambda = 0.0005$. The same system was used for Gurobi and ABS2. Although BQP achieves a better final objective value, the QUBO approach reaches good solutions in a substantially shorter time–without developing specialized heuristics.

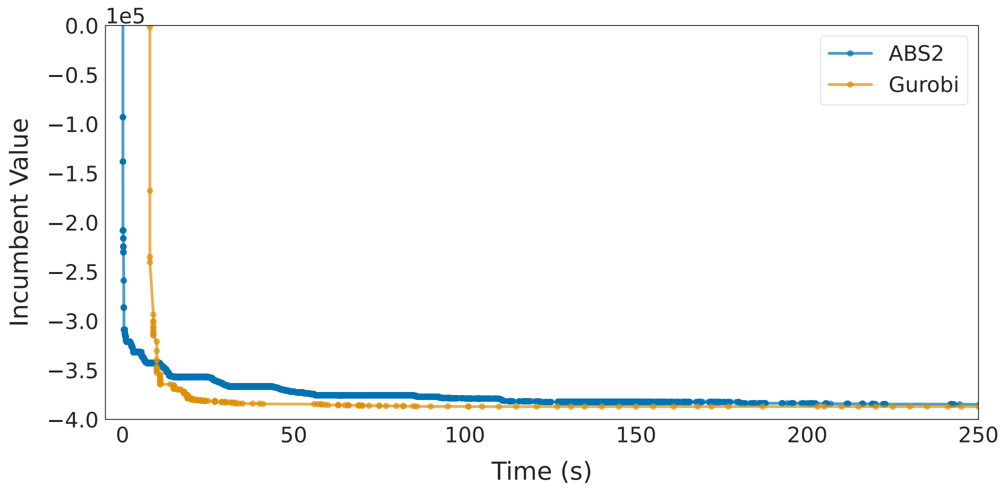


Figure 12: (Portfolio) The plot illustrates the best found objective values at every step of the solving for different risk aversion parameters with Abs2 and Gurobi solvers for one instance with $\lambda = 0.0005$ and 50 assets. While ABS2 finds good solutions faster, Gurobi eventually achieves better results.

4.7 Maximum Independent Set

4.7.1 Background

An independent (or stable) set is a set of vertices not adjacent to each other in a graph, i.e., no edge connects any two vertices of the set. The maximum independent set (MIS) problem (or maximum stable set problem) is to find the largest independent set for a given graph. It is deeply rooted in graph theory and combinatorial optimization and has a long history of research. Its decision problem is one of the first problems shown to be **NP**-complete [Kar72]. It is known to be strongly **NP**-hard for general graphs, while it can be solved in polynomial time for certain classes of graphs, e.g., P_5-free graphs and perfect graphs. The independent set is a fundamental feature of a graph and relates to various other graph parameters. The MIS for a graph is equivalent to the maximum clique problem for its complement graph. A set of independent vertices is equivalent to its complement set being a vertex cover. Thus, the sum of the size of the MIS and the size of the minimum vertex cover is equal to the number of vertices in the graph. The MIS is also useful for modeling industry problems, e.g., non-intersecting label placement on a map [AVKS98], discovering stable genetic components for designing engineered genetic systems [HLH+20], and strategic planning, such as choosing locations for stores [WLG+22].

Why is it interesting?

The unweighted maximum independent set problem is a classic **NP**-hard mathematical problem. Starting at problem sizes of several hundred variables, there are known instances that are hard to solve to proven optimality with existing methods [Slo25], and which are difficult to tackle even heuristically. An interesting property of this problem class is that it is well-suited for translation into relatively sparse QUBOs–often with small coefficients.

There have been extensive studies both on exact and approximate algorithms. As far as we know, the best-known exact algorithm can compute the MIS of an n-vertex graph in $1.1996^n n^{O(1)}$ time and polynomial space [XN17]. As for approximate algorithms, most studies (too many to list here) consider a particular class of graphs and develop algorithms with constant approximation ratios since the maximum independent set problem, in general, is Poly-APX-complete, meaning it is as hard as any problem that can be approximated to a polynomial factor [BEP05]. Quantum algorithms have also been investigated recently. For example, a method is proposed for solving the maximum independent set problem on unit disk graphs [CCJ90] using a quantum simulator with Rydberg atomic arrays [EKC+22]. It should be noted that for unit disk graphs, better approximation results are available than for the general case; see, e.g., [MBHI+95, DdJ20].

4.7.2 Problem statement

Let $G = (V, E)$ be a graph with vertices $V = \{1, \ldots, n\}$ and edges $E \subset V \times V$. The maximum independent set problem for the graph G is

$$\max_{S \subseteq V} |S| \text{ with } i, j \in S \Rightarrow (i, j) \notin E.$$

It can be represented as a Binary Linear Program (BLP):

$$\max_{x \in \{0,1\}^n} \sum_{i \in V} x_i \text{ subject to } x_i + x_j \leq 1 \text{ for all } (i, j) \in E,$$

or as a Binary Quadratic Program (BQP):

$$\max_{x \in \{0,1\}^n} \sum_{i \in V} x_i^2 \text{ subject to } x_i x_j = 0 \text{ for all } (i, j) \in E.$$

Note that $x_i = x_i^2$ holds for binary variables. The constraint in each formulation forbids $x_i = x_j = 1$, for all $(i, j) \in E$, i.e., both ends of each edge cannot be in an independent set. For the weighted version, some weights $w_i > 0$, $i \in V$, can be added to the vertices.

4.7.3 Instances

Any collection of graphs might be a suitable benchmark set for MIS. We aimed to collect some instances known to be challenging and added several smaller ones to facilitate progress tracking. We collected instances from various sources, in particular: Network Repository [RA15], SteinLib [KMV00], OESIS Implementation Challenge [Slo25], and self-generated graphs. Furthermore, we provide a program that takes a graph and an independent set and verifies that it is indeed a stable set in the graph.

4.7.4 Classical Baseline

We provide formulations as binary linear programs and as QUBO. These can be solved by any suitable solver. We provide baseline results for the binary linear program using Gurobi 12.0.1 run on an Intel Xeon Gold 6338 CPU @ 2.00 GHz processor using 128 threads with a timeout of 7200 seconds. The baseline results for the QUBO formulation are computed using ABS2 as a heuristic run on four Nvidia A40 48GB with a timeout of 7200 seconds. Corresponding runtime, TTS, and absolute gaps to the optimal/best-known solutions can be found in Figure 13. Exact data and instance names can be obtained from Table 7.

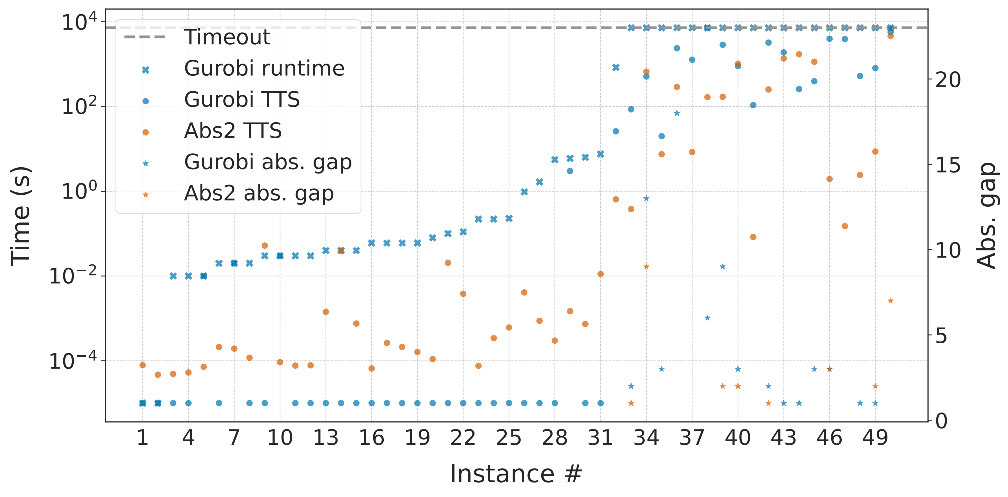


Figure 13: (Independent Set) Runtimes, time-to-solution, and absolute gap to optimal/best-known solution for the problem sorted by Gurobi's runtime. Only gaps > 0 are plotted.

						Gurobi		Abs2		
#	Name	Nodes	Edges	Best	RT (s)	TTS (s)	Gap	TTS (s)	Gap	Source
4	farm	17	39	10	0.0	0.0	0	0.0	0	[RA15]
2	mammalia-kangaroo-interactions	17	91	4	0.0	0.0	0	0.0	0	[RA15]
3	johnson8-2-4	28	210	7	0.0	0.0	0	0.0	0	[RA15]
8	ibm32	32	90	13	0.0	0.0	0	0.0	0	[RA15]
5	karate	34	78	20	0.0	0.0	0	0.0	0	[RA15]
1	football	35	118	16	0.0	0.0	0	0.0	0	[RA15]
7	chesapeake	39	170	17	0.0	0.0	0	0.0	0	[RA15]
16	MANN-a9	45	918	3	0.1	0.0	0	0.0	0	[RA15]
10	aves-sparrow-social	52	454	13	0.0	0.0	0	0.0	0	[RA15]
11	insecta-ant-colony1-day38	56	1134	6	0.0	0.0	0	0.0	0	[RA15]
6	sloane_1dc_64	64	543	10	0.0	0.0	0	0.0	0	[RA15]
23	hamming6-4	64	704	12	0.2	0.0	0	0.0	0	[Slo25]
12	hamming6-2	64	1824	2	0.0	0.0	0	0.0	0	[RA15]
20	johnson8-4-4	70	1855	5	0.1	0.0	0	0.0	0	[KMV00]
13	es60fst03	113	142	55	0.0	0.0	0	0.0	0	[RA15]
19	johnson16-2-4	120	5460	15	0.1	0.0	0	0.0	0	[RA15]
15	es60fst01	123	159	60	0.0	0.0	0	0.0	0	[RA15]
26	C125-9	125	787	34	1.0	0.0	0	0.0	0	[Slo25]
24	sloane_1zc_128	128	1120	18	0.2	0.0	0	0.0	0	[Slo25]
22	sloane_1dc_128	128	1471	16	0.1	0.0	0	0.0	0	[KMV00]
18	sloane_2dc_128	128	5173	5	0.1	0.0	0	0.0	0	[KMV00]
25	insecta-ant-colony3-day09	160	8883	9	0.2	0.0	0	0.0	0	[Slo25]
9	es60fst04	162	238	78	0.0	0.0	0	0.1	0	[RA15]
27	keller4	171	5100	11	1.7	0.0	0	0.0	0	[RA15]
14	es60fst02	186	280	88	0.0	0.0	0	0.0	0	[RA15]
17	c-fat200-1	200	1534	18	0.1	0.0	0	0.0	0	[RA15]
21	gen200_p0-9_44	200	1990	44	0.1	0.0	0	0.0	0	[KMV00]
29	brock200-2	200	10024	12	6.0	3.0	0	0.0	0	[RA15]
31	brock200-3	200	12048	9	7.6	0.0	0	0.0	0	[RA15]
30	brock200-4	200	13089	8	6.3	0.0	0	0.0	0	[RA15]
28	brock200-1	200	14834	6	5.6	0.0	0	0.0	0	[RA15]
37	brock400-1	400	20077	27	7200.1	1269.0	0	8.4	0	[RA15]
33	R_500_005_1	500	6256	91	7200.0	86.0	2	0.4	1	[RA15]
35	C500-9	500	12418	57	7200.1	20.0	3	7.5	0	[Riv24]
45	brock800-1	800	112095	23	7200.3	395.0	3	1139.3	0	[RA15]
49	frb45-21-3	945	58245	45	7201.5	806.0	1	8.7	2	[RA15]
39	R_1000_005_1	1000	24670	117	7200.1	2859.0	9	169.6	2	[RA15]
41	hamming10-4	1024	89600	40	7200.2	108.0	0	0.1	0	[GHG+21]
42	frb50-23-3	1150	81068	50	7200.2	3226.0	2	252.3	1	[RA15]
40	frb53-24-1	1272	94227	53	7200.1	903.0	3	1023.5	2	[RA15]
32	socfb-haverford76	1446	59589	282	838.7	26.0	0	0.6	0	[RA15]
47	p_hat1500-3	1500	277006	94	7200.8	3915.0	0	0.2	0	[RA15]
48	p_hat1500-1	1500	839327	12	7201.4	523.0	1	2.4	0	[RA15]
46	frb59-26-2	1534	126163	59	7200.8	3980.0	3	1.9	3	[RA15]
34	sorrell7	2048	39424	198	7200.1	513.0	13	663.5	9	[RA15]
44	sorrell4	2048	504451	24	7200.2	257.0	1	1708.8	0	[RA15]
43	socfb-trinity100	2613	111996	499	7200.2	1891.0	1	1361.5	0	[RA15]
36	keller6	3361	1026582	59	7200.1	2377.0	18	290.9	0	[Riv24]
50	frb100-40	4000	572774	100	7205.1	5695.0	23	4634.8	7	[RA15]
38	C4000-5	4000	3997732	18	7200.1	7200.1	6	167.0	0	[RA15]

Table 7: (Independent Set) Best solution, Runtimes, TTS, and absolute gaps to best found (some even optimal) solutions for Gurobi 12.0.1 and Abs2. Notably, only Gurobi checks optimality. Abs2 runs as a heuristic. Please note that we do not report any runtimes for ABS2, since the stopping criteria for these heuristic solvers vary.

4.8 Network Design

4.8.1 Background

Network design models have wide applications in telecommunications, transportation planning [Gav91, MW84], and power networks [SFE+23]. One common formulation for this problem is to use an MIP formulation [BG95]. Given an $n \times n$ matrix T, where each entry t_{ij} represents the traffic to be routed between vertex i and j, and an integer $p > 0$, construct a simple directed graph $D = (V, A)$ with vertex set $V = \{1, \ldots, n\}$ and a subset of selected edges $S^* \subseteq A$, where each vertex has indegree and outdegree equal to p. Furthermore, all the demands are routed such that the maximum load on any single edge of the network is minimized. The load on an edge is the sum of all the traffic that passes through that edge. The problem is **NP**-hard [BG95].

Why is it interesting?

The min-max objective function underlying network design models is linear, and the problem often has a sparse structure. While it is trivial to construct a feasible solution, finding optimal solutions is very difficult. The particular problem presented in this section has been withstanding all attempts to solve it to proven optimality for the last 30 years in its 24-node size. Furthermore, network design models have wide applications in telecommunications and transportation planning [Gav91, MW84].

Although we will focus on the mathematical formulation as a multicommodity-directed model, many other formulations have been used in network design, depending on the problem context. In [GCF99, ABW02], the authors provide surveys of alternative formulations. In the standard formulation, both flow decision variables and design decision variables are modeled explicitly [BG95, BCGT98, GCF99]. In this case, the linear programming relaxations of multicommodity flow formulations provide lower bounds that are normally weak [GCF99, ABW02]. To overcome this issue, in [ABW02], the authors remove the flow variables as explicit decisions and embed them within the design variables. Furthermore, in [GCF99, ABW02] the authors also provide a survey of different methods that have been used for network design. For example, in [MMW86], the authors apply Benders decomposition and demonstrate the effectiveness of variable elimination preprocessing and a dual ascent procedure to accelerate the decomposition algorithm. In [BG95], the authors report on computational experience with a cutting plane algorithm for the above formulation. In [BS96], they use the column generation technique to obtain near-optimal solutions.

4.8.2 Problem statement

Given a matrix $T \in \mathbb{N}^{n \times n}$, and an integer $p > 0$, construct a simple directed graph $D = (V, A)$ with vertex set $V = \{1, \dots, n\}$, where each vertex has indegree and outdegree equal to p, and simultaneously, in D, route t_{ij} units of flow from vertex i to j, for all $1 \leq i \neq j \leq n$, such as to minimize the maximum aggregated flow on any arc in D.

We can formulate this network design problem as a multicommodity design problem as follows: For $k, i, j \in \{1, \dots, n\}$ we define binary variables $x_{ij} \in \{0, 1\}$ which are 1 iff an arc is established between vertex i and j and 0, otherwise. Further, we define non-negative integer variables $f_{kij} \in \mathbb{Z}_+$ that represent the flow of commodity k between vertices i and j, and a suitable big constant M. Then we get:

$$
\begin{aligned}
&\min_{x,f,z} z && k, i, j \in \{1, \dots, n\} && \\
&\text{subject to} \sum_{j \neq i} x_{ij} = p && \text{for all } i\,, && (5)\\
&\sum_{j \neq i} x_{ji} = p && \text{for all } i\,, && (6)\\
&f_{kij} \leq M x_{ij} && \text{for all } k, i \neq j\,, && (7)\\
&\sum_{j \neq i} f_{kij} - \sum_{j \neq i} f_{kji} = t_{ki} && \text{for all } i, k\,, && (8)\\
&\sum_{k} f_{kij} \leq z && \text{for all } i \neq j\,. && (9)
\end{aligned}
$$

Constraints (5) and (6) set the outdegree and indegree of each vertex to p. Constraint (7) ensures that flow is only allowed on arcs in A. Constraint (8) ensures the flow conservation for each commodity k. (9) ensures z is greater than the aggregated flow on any arc. Since the objective is to minimize z, (9) will be met with equality at the arc with the highest aggregated flow.

4.8.3 Instances

The original problem by [BG95] set $p = 2$, $n = 24$, $T \in \{0, \dots, 100\}^{n \times n}$, and had continuous flow variables. We have multiplied all demands by 1,000 and require an integral flow, making the problem completely integer. To provide smaller versions of the problem, the benchmark instances use the data of the first n vertices from $n = 5$ to 24.

We provide a checking routine that, given a sequence n, a file with the demands, and a file with a solution, checks whether the solution is feasible and computes the objective value.

4.8.4 Classical Baseline

We provide a basic IP formulation of the problem. More sophisticated cutting planes are described in [BG95]. Results with the basic IP formulation found using Gurobi 11.0.0 on an AMD EPYC-7542 32-core processor with 64 threads and a time limit of 2 hours are summarized in Figure 14. We were able to solve the problem up to size $n = 9$ to proven optimality within the time limit. For larger sizes, heuristic, but not necessarily optimal, solutions are provided. The instance with 24 vertices and continuous flow variables is also part of MIPLIB2017 [GHG+21], the latest results of which can be found at [dan]. Note that those results correspond to the unscaled model version with continuous variables, which, therefore, has a slightly different objective value than the scaled one. However, the solutions are compatible.

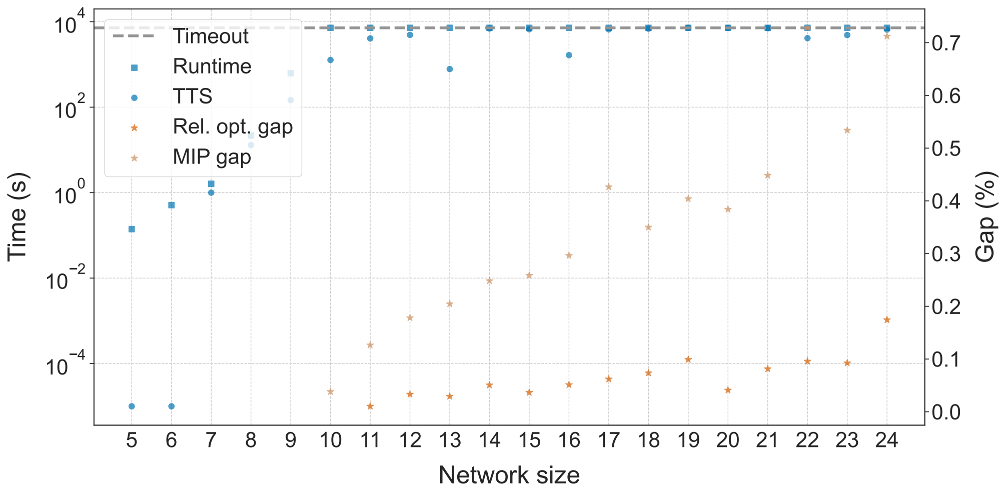


Figure 14: (Network) *Runtime* is the time to compute a proven optimal solution. *Time-to-solution* is the time until the best solution was found. *Rel. opt. Gap* is the gap to the optimal or best-known solution. *MIP Gap* is the gap to the lower bound. Only gaps > 0 are plotted.

4.9 Vehicle Routing Problem

4.9.1 Background

The Vehicle Routing Problem (VRP) is the problem of determining the most efficient routes for a fleet of vehicles to deliver goods or services to a set of customers, typically starting and ending at a central depot. This problem is highly relevant for many real-world applications in logistics, transportation, and supply chain management, see, e.g., [CHH22]. Originally proposed in [DR59] as 'the truck dispatching problem', many different variants of VRP have emerged since then [Hel17, VLM20]. Providing a collection of relevant and difficult VRP instances is a complex task because there are numerous variants. Although classical heuristics for VRP are practically very successful, many papers on quantum approaches for VRP problems have been published [XLC+24, MBF23, LDTA23, FGL+21]. We include the VRP problem class in this collection to facilitate a comparison with existing work and to explore the capabilities of quantum optimization algorithms for a practically relevant problem.

We limit ourselves to a specific variant of VRP, the Capacitated Vehicle Routing Problem (CVRP), where the objective is to minimize the aggregated costs across all routes while satisfying the vehicle capacity, effectively limiting the number of goods that each vehicle can carry. This variant, in fact, corresponds to the original problem formulation from [DR59], where it was presented as a generalization of the TSP [MTZ60]. Compared to TSP, VRP involves an additional level of decision-making, as customers not only need to be served but also assigned to vehicles. An example CVRP instance and its optimal solution are visualized in Figure 15.

Why is it interesting?

Determining the optimal solution to a Vehicle Routing Problem (VRP) is **NP**-hard [LK81] and of high practical relevance as it has many direct industrial applications and, hence, the potential for a significant business value impact. Similar to the network problem from the previous section, it is trivial to find feasible (but not necessarily optimal) solutions. Due to the complexity of the problem, mostly small instances can be solved exactly, but heuristics and metaheuristics are often well-suited to tackle real-world instances. This problem class has already received significant attention from the quantum community.

Due to the practical and academic relevance of the CVRP, numerous optimization approaches have been developed, and research continues [SM21]. Exact methods include branch-and-bound and branch-and-cut strategies. A well-known heuristic is the *savings algorithm* [CW64], which has been modified and improved in various ways. CVRPs can be further divided into symmetric CVRPs and asymmetric CVRPs, depending on whether the direction of a tour affects its cost. Some solution approaches can only be applied to symmetric CVRPs. A review of established exact and heuristic methods can be found in [TV02a, TV02b]. Metaheuristics include genetic algorithms, simulated annealing, and Tabu search. A recent comparison of different heuristics and metaheuristics can be found in [MDM24]. Alternative research directions are approaches based on machine learning [EJBB24]. More recently, quantum optimization has been explored as a potential strategy. Since naive QUBO formulations for CVRP require many variables [HNN+20], hybrid strategies have been used, for example, by separating the problem into a clustering and routing phase [FRG+19, HMCL24]. Many hybrid approaches rely on solving the vehicle-customer assignment classically and then solving TSPs using quantum optimization. Successful applications will need to overcome this simplification, as solving TSPs exactly, e.g., via Concorde [ABCC03], is unlikely to be outperformed by quantum methods anytime soon.

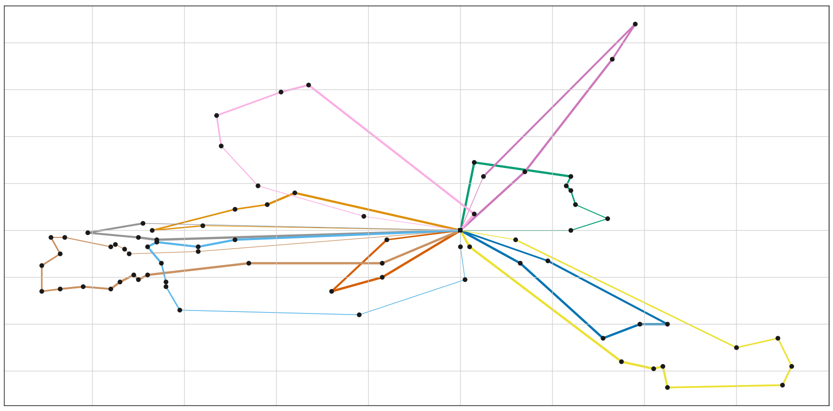


Figure 15: (Routing) Example instance *tai75a* (75 customers, 10 vehicles) from the Rochat and Taillard dataset [RT95] and optimal solution as obtained from CVRPLIB [LOQ]. Each line color represents the route of one vehicle between the customers (●) starting and ending at the central depot (■). The thickness of a line indicates the remaining capacity of the corresponding vehicle along the route. This example demonstrates the combinatorial complexity of finding the best route among all possible routes while fulfilling capacity constraints.

4.9.2 Problem statement

Consider a CVRP with $n \geq 1$ customers, where each customer i has a demand $d_i \in \mathbb{N}$ with $i \in \{1, \ldots, n\}$. Furthermore, we assume that there is a homogeneous fleet of $K \geq 1$ vehicles, where each vehicle has a total capacity of $Q \in \mathbb{N}$. Each vehicle starts and ends its tour at the depot. The cost of moving a vehicle from customer i to j is given by $c_{ij} \in \mathbb{N}$ with $i, j \in \{0, \ldots, n\}$ and $i \neq j$, where "customer 0" is used to represent the depot. With this definition, we presume all-to-all connectivity between the customers (and the depot). While the cost matrix is generally arbitrary, we focus here on a special case relevant to the benchmarking instances we provide, where the costs are based on graph vertex distances in a metric space that satisfies non-negativity (as given by the cost domain) and symmetry (i.e., $c_{ij} = c_{ji} \, \forall \, i, j \in \{0, \ldots, n\}$, the condition for the aforementioned symmetric CVRP), which means they also obey the triangle inequality (i.e., $c_{ij} \leq c_{ik} + c_{kj} \, \forall \, i, j, k \in \{0, \ldots, n\}$). Each customer must be served exactly once by a vehicle, while the total customer demand on a given route must not exceed the capacity limit of the vehicle. The optimization goal is to determine a feasible solution that minimizes the aggregated costs across all routes. This does not necessarily require the use of all K vehicles, i.e., some may remain in the depot and, hence, do not contribute to the costs.

A CVRP can be modeled in many different ways, one of which is the two-index vehicle flow formulation with Miller-Tucker-Zemlin (MTZ) constraints [SM21]. Following [MDS17], this formulation reads:

$$\min_{x,y} \sum_{\substack{i,j=0\\ i\neq j}}^{n+1} c_{ij}x_{ij} \tag{10}$$

$$\text{subject to} \sum_{\substack{j=1\\ j\neq i}}^{n+1} x_{ij} = 1 \qquad \text{for all } i \in \{1\ldots,n\}\,, \tag{11}$$

$$\sum_{\substack{i=0\\ i\neq h}}^{n} x_{ih} = \sum_{\substack{j=1\\ j\neq h}}^{n+1} x_{hj} \qquad \text{for all } h \in \{1,\ldots,n\}\,, \tag{12}$$

$$\sum_{j=1}^{n} x_{0j} \leq K\,, \tag{13}$$

$$y_j \geq y_i + d_j x_{ij} - Q(1-x_{ij}) \qquad \text{for all } i \neq j \in \{0,\ldots,n+1\}\,, \tag{14}$$

$$d_i \leq y_i \leq Q, \qquad \text{for all } i \in \{0,\ldots,n+1\}\,, \tag{15}$$

$$x_{ij} \in \{0,1\} \qquad \text{for all } i,j \in \{0,\ldots,n+1\}\,, \tag{16}$$

$$y_i \in \mathbb{N}_0 \qquad \text{for all } i \in \{0,\ldots,n+1\}\,. \tag{17}$$

In this IP, the optimization is done with respect to two types of decision variables. First, the binary variables $x \in \{0,1\}^{(n+2)\times(n+2)}$ and second, the integer variables $y \in \mathbb{N}_0^{n+2}$. The binary variable x_{ij} takes the value 1 if and only if there is a direct route between customers i and j for $i,j \in \{0,\ldots,n+1\}$, where "customer 0" is the depot, as introduced above. To simplify the notation, we also symbolically introduce "customer $n+1$" to represent the same depot, which allows us to distinguish between starting at the depot ($i=0$) and ending at the depot ($i=n+1$), where $c_{j,n+1} := c_{0j}$ for all $j \in \{1,\ldots,n\}$. Furthermore, the integer variable y_i represents the cumulated demand on the route that visits customer i for $i \in \{0,\ldots,n+1\}$, where $d_0 := d_{n+1} := 0$.

The objective, Eq. (10), represents the aggregated costs of all routes that need to be minimized. The first set of constraints, Eq. (11), ensures that each customer is only visited once. The second set of constraints, Eq. (12), ensures a correct flow of vehicles. The third constraint, Ineq. (13), ensures that only up to K vehicles can leave the depot. The fourth and fifth set of constraints, Ineq. (14)) and (15), follow the MTZ modeling of a TSP [MTZ60] and are responsible for ensuring compliance with the vehicle capacities. Using this formulation, a polynomial number of capacity constraints is sufficient (in contrast to an exponential number of constraints with alternative models). Finally, Eq. (16) and (17) are the integrality constraints.

The problem is often also formulated with real-valued costs and real-valued demands, which only requires the change of y from an integer variable to a continuous variable $y \in \mathbb{R}$, effectively leading to a mixed-integer program. For a more convenient benchmarking setup, we only provide the integer-based formulation, since it lets us avoid having to consider machine precision for the instance data, while simultaneously keeping the problem complexity the same.

4.9.3 Instances

A survey on benchmark data sets can be found in [GKM+21]. All instances in the repository are given for $k=4$ vehicles and $n=20$ customers and were created to be 100% tight, i.e., the sum of the demands equals the vehicle capacity Q. The instances follow the TSPLIB/CVRPLIB format [LOQ]. We randomly generated and solved several thousand instances, selecting those that took the longest to solve, which is still only seconds on modern digital systems. Out of the set of problem sizes considered, the chosen instances are among the most difficult. It has proven unexpectedly hard to find small-to-medium instances that are primal difficult. Classical methods solve these sizes quickly, see [UPP+17, GKM+21]. This indicates that quantum approaches would have to be extremely fast to be competitive on these problems in the near term. We plan to add and investigate larger instances in the future.

4.9.4 Classical Baseline

It should be noted that numerous open-source codes for VRP are available, e.g., [LKH, goo, HGS], with some even utilizing GPU acceleration [cuO]. Additionally, several companies offer specialized, highly optimized solvers for this problem; however, it is difficult to assess their performance. Furthermore, they are tailored to real-world scenarios, which, for example, include multiple vehicle classes and driver break times.

Table 8 shows the optimal cost and the average time taken by HGS-CVRP [HGS, Vid22] to find this solution, measured over 10 runs with different seeds and a time limit of 10 seconds. Notably, the optimal solution could not be found within this limit, for instance, 21. However, given that HGS-CVRP takes only a few milliseconds to find an optimal solution for typical instances with $n = 20$, we expect it to be found within a marginally longer time limit. The optimal solutions were certified using VRPSolver [EQSU23, PSUV20, VRPa, VRPb].

No	Cost	Time (s)
01	646	1.74
02	650	0.33
03	508	1.30
04	776	1.64
05	702	1.01
06	690	0.85
07	730	0.82
08	718	0.65
09	707	0.88
10	737	0.61
11	914	4.48
12	709	1.18
13	628	0.47
14	696	1.44
15	780	0.94
16	830	3.55
17	605	1.03
18	997	1.07
19	976	2.10
20	648	8.71
21	842	>10.00
22	760	0.98
23	851	2.84
24	725	0.82
25	462	0.49
26	688	1.01
27	540	1.40
28	669	2.27
29	658	0.94
30	685	0.78
31	445	7.16
32	882	2.38
33	568	0.60
34	891	8.51
35	707	1.18
36	610	2.03
37	650	0.81
38	991	0.98
39	717	5.23
40	700	2.16
41	736	1.33
42	998	1.30
43	465	1.95
44	973	1.11
45	1071	5.10
46	812	0.79
47	583	0.54
48	544	1.81
49	1045	0.56
50	627	0.89
51	491	0.91
52	622	2.35
53	967	7.12
54	791	2.12
55	617	3.73

Table 8: CVRP: Optimal cost values and running times.

4.10 Topology Design

4.10.1 Background

Why is it interesting?

Several GraphGolf competitions [KFH+15] over the years have shown that specifically the Order Degree Problem (ODP) is difficult to solve in practice–although the resulting problem formulation can be relatively sparse, and a problem instance is specified by providing just three integers. Furthermore, the variables are typically binary, which facilitates the encoding of the problem in a quantum system. The problem finds application in computer network design [SWS11].

Given a graph $G = (V, E)$, we define three possibly competing objectives. First, one may want to focus on *order* such that the number of vertices $n = |V|$ is maximized. Second, one can target the *degree*, which relates to the minimization of the maximum vertex degree $d = \max_{v \in V} \delta(v)$. Lastly, one may want to minimize the *diameter* $k = \max_{u,v \in V} \text{length\_of\_shortest\_path}(u, v)$. Fixing two parameters within the set {order, degree, diameter}, and optimizing the third, results in the following optimization problems:

- Order Degree Problem (ODP): minimize the diameter;
- Degree Diameter Problem (DDP): maximize the number of vertices;

- Order Diameter Problem (ODiP): minimize the maximum vertex degree.

Notably, the name of the problem is given by the two fixed parameters. The ODiP has received little attention to date. The DDP has been considered extensively, with the foundations discussed already in the literature from the 1960s [HS60, CCMS74], and further explored in the 1980s, e.g., [Chu86, Chu87]. For a comprehensive survey, we refer the reader to [MS12]. In this work, we consider the ODP, for which–to our knowledge–an **NP**-hardness proof is not available in the open literature. The ODP arises in the search for network topologies with low latency [SWS11], and a variety of methodologies for constructing networks with low latency are discussed in the literature [BH14, SHPG12]. We use the ODP variant employed in the GraphGolf competitions [KFH+15]: the goal is to solve the ODP, using the average shortest path length (ASPL) as a tie-breaker for graphs with the same diameter, with lower ASPL being preferred. This problem is difficult to solve in practice [KFH+15].

4.10.2 Problem statement

Let $\Delta(s,t)$, $s,t \in V$ be the shortest distance of two vertices in a simple unweighted graph $G = (V,E)$. Given a number $n \in \mathbb{N}$ of vertices, and a maximum vertex degree $d \in \mathbb{N}$. We are looking for a simple graph $G = (V,E)$, $V = \{1,\dots,n\}$, $E \subseteq \{(u,v) \in V \times V | u \neq v\}$ with $\delta(v) \leq d$, for all $v \in V$, that minimizes $\max_{s,t \in V} \Delta(s,t)$. Multiple formulations for the ODP are possible; for an overview, we refer to [Wan21]. There is considerable freedom in the choice of how the lengths of the all-pairs shortest paths are computed.

n	d	k
15	4	2*
15	3	3*
20	3	3*
20	4	3*
20	5	2*
25	3	4*
25	4	3*
25	5	3*

n	d	k
25	6	2*
30	4	3*
30	5	3*
30	6	3
35	5	4
35	6	3
40	6	3*
50	4	5

n	d	k	**Submitter**
512	4	6	EvbCFfp1XB
512	6	5	MasakiChujo
1024	4	7	EvbCFfp1XB
1726	30	3	(random)
4855	15	4	Teruaki Kitasuka
9344	6	7	EvbCFfp1XB
65536	6	9	(random)
100000	8	7	Hajime Terao
1000000	16	6	Teruaki Kitasuka
1000000	32	5	(random)

Table 9: (Topology) Overview of the provided instances and their optimal/best-known solutions. Proven optimality is indicated by $*$. The solutions shown in the left and center tables were computed by us using Gurobi 11.0.0. and result submissions shown in the right table can be found at [KFH+15].

4.10.3 Instances

The instances range from 15 vertices and degree 3 to 10^6 vertices and degree 32. A detailed presentation of the provided instances can be found in Table 9. We provided a program to check solutions, i.e., given a triple (n,d,k) and a graph, it verifies that the graph has n nodes, a maximum degree of d, and a diameter of at most k.

4.10.4 Classical Baseline

Our baseline was computed using Gurobi 11.0.0 on an AMD EPYC-7542 32-core processor using 64 threads with a time limit of 7200s. In the repository, we provide three different MIP formulations. The first model uses a standard s-t flow model to compute shortest path values. The second model uses Seidel's quadratic all-pairs shortest path formulation. The third model is a linearization of Seidel's quadratic model. This problem is hard to model as a compact integer program. Gaps and runtimes can be found in Figure 16. The results indicate that we can prove optimality only on rather small instances if we use an MIP solver. With the provided models, even solving $n = 30, d = 6$ takes too long. However, as can be seen on [KFH+15], using heuristic methods, it is possible to compute solutions to larger instances.

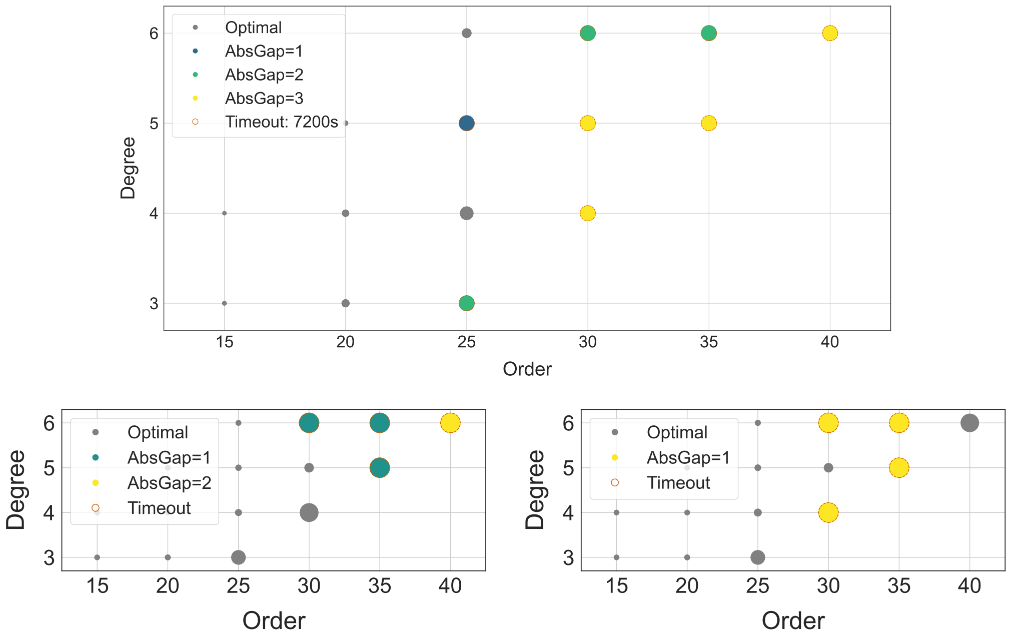


Figure 16: (Topology) Absolute MIP gap corresponding to order and degree of the *flow* (top) / *quadratic* (bottom left) and *linearized* (bottom right) Seidel model. The size of the points indicates the running time–the bigger, the longer. Furthermore, the timeout is set to 7200s.

5 Submission Guidelines and Illustrative Results

The following section presents guidelines for submitting benchmark results to [QOBLIB] and provides exemplary baseline results for three selected problem classes–*Low Autocorrelation Binary Sequences*, *Minimum Birkhoff Decomposition* and *Independent Set* problem–each solved with another quantum optimization algorithm. These results can also be found in the [QOBLIB] repository.

We provide a submission template in Table 10 which should be submitted together with the discovered solution. The metrics were chosen to enable a fair and systematic comparison of solutions and algorithms. More specifically, we ask for the identification of the concrete problem instance and the submitter(s). Given that not every detail can be given in a compact form, submissions ideally refer to a paper or a code repository with further information, such as hyperparameters, additional hardware specifications, software versions, etc. Furthermore, the best objective value found (for optimization problems) as well as the corresponding solution (in addition to the template) should be given. If supported by an algorithm, an a posteriori bound on the optimal objective value can be provided. Next, the chosen modeling approach should be described (e.g., QUBO, ILP, etc.) as well as the resulting number and type of decision variables and (non-zero) coefficients needed to represent the considered problem instance. Furthermore, the submission should provide a brief summary of the complete optimization workflow to facilitate reproducibility. This includes pre-processing, pre-solver, main optimization algorithm, and post-processing, as well as an indication if the algorithm is deterministic or stochastic. Stochastic algorithms are generally recommended to be repeated multiple times. In this case, the number of successful runs that result in feasible solutions or return solutions close to the best found solutions (cf. Table 10 for more details) should be reported. Lastly, the overall runtime and the runtime spent on the various hardware platforms should be provided. In case of multiple repetitions, the average runtime over all repetitions may be reported. Additional information, such as the distribution of the runtimes or potential correlations with solution quality, is encouraged to be described in a corresponding publication or reference. As discussed in Section 3, the runtimes should not include potential queuing times for hardware access. For most problems, we provide solution checkers in [QOBLIB]. These should be used to verify proposed solutions before a benchmark result is submitted to [QOBLIB].

5.1 Low Autocorrelation Binary Sequences

In the following, we focus on tackling the LABS problem using the bias-field digitized counter-diabatic quantum optimization (BF-DCQO) algorithm [isk]. The respective results are compared with QAOA and various classical optimization algorithms.

Problem Identifier	Identifier of the considered problem instance.
Submitter	Name(s) of the submitter(s) and affiliation(s).
Date	Date of submission.
Reference	Reference to a paper/repository with more details.
Best Objective Value	The best objective value found by the algorithm across all repetitions.
Optimality Bound	Lower bound (minimization) or upper bound (maximization) for the optimal objective value, if supported; otherwise, set to N/A.
Modeling Approach	Describe how the considered problem instance is modeled.
#Decision Variables	Total number of decision variables.
#Binary Variables	Number of binary decision variables.
#Integer Variables	Number of integer decision variables.
#Continuous Variables	Number of continuous decision variables.
#Non-Zero Coefficients	Number of non-zero coefficients in objective function and constraints.
Coefficients Type	Type of coefficients, such as integer, binary, continuous.
Coefficients Range	Range of non-zero coefficients, i.e., min/max values.
Workflow	Description of the optimization workflow: pre-processing, pre-solvers, machine learning training, optimization algorithms, post-processing, etc.
Algorithm Type	Indicate whether the algorithm is deterministic or stochastic.
#Runs	The number of times the experiment has been repeated.
#Feasible Runs	The number of times a run found a feasible solution.
#Successful Runs	Number of runs that found a feasible solution with objective value $\leq (1+\epsilon) * f_{\min}$ (minimization) or $\geq (1-\epsilon) * f_{\max}$ (maximization), where $f_{\min/\max}$ is the best solution found by the algorithm.
Success Threshold	The threshold ϵ to define a successful run.
Hardware Specifications	Specifications of hardware used to run the workflow.
Total Runtime (Wall-clock)	Total runtime to run the complete workflow.
CPU Runtime	CPU runtime to run the workflow.
GPU Runtime	GPU runtime to run the workflow.
QPU Runtime	QPU runtime to run the workflow.
Other HW Runtime	Runtime on other hardware to run the workflow.

Table 10: The table summarizes the metrics that should be reported when submitting benchmark results to [QOBLIB]. All *runtimes should be reported as average* if multiple algorithm runs were executed.

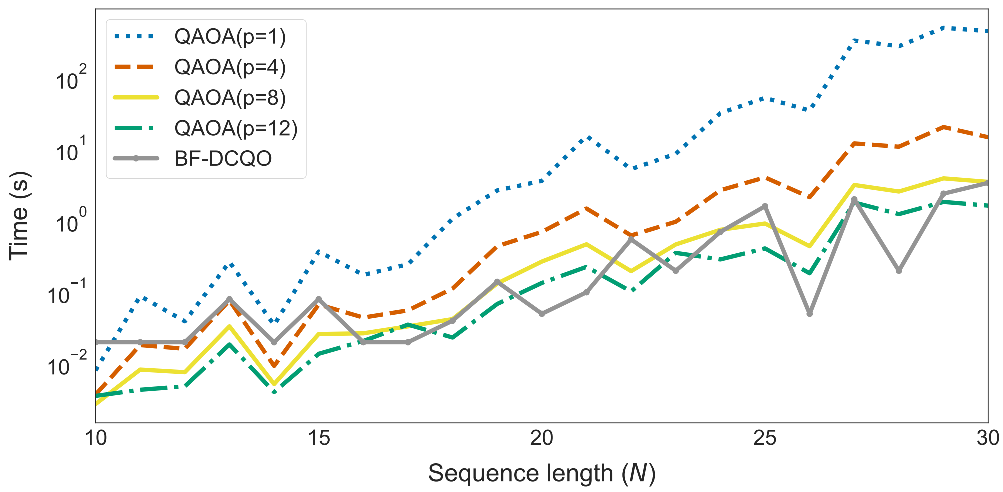


Figure 17: (LABS) Comparison of the time-to-solution for LABS instance ranging from size 10 to 30 using BF-DCQO and QAOA across different numbers of layers: $p = 1$, $p = 4$, $p = 8$, and $p = 12$. The data for QAOA is sourced from [SLC+24]. Note that the BF-DCQO times reflect idealized QPU runtime estimates, excluding transpilation.

The quantum adiabatic algorithm (QAA) evolves an easily prepared ground state of an initial Hamiltonian H_i to the unknown ground state of a target Hamiltonian H_f, which should correspond to the optimal solution of an optimization problem of interest. This evolution is described by a time-dependent Hamiltonian of the form

$$H_{\mathrm{ad}}(t) = [1-\lambda(t)]H_i + \lambda(t)H_f\ , \tag{18}$$

where $\lambda(t) \in [0,1]$ guides the adiabatic path. The corresponding time evolution can be discretized and run on a digital quantum computer. In QAA, the system remains in the instantaneous ground state in the limit $\partial_t\lambda(t) \to 0$. In the digitized counterdiabatic quantum optimization (DCQO) protocol counterdiabatic terms are added to the time-dependent Hamiltonian of Eq. (18) [HPD+21, CHP+22, HCS22], which becomes

$$H_{\mathrm{cd}} = H_{\mathrm{ad}}(t) + \partial_t\lambda(t)A_\lambda\ .$$

Here, A_λ is the adiabatic gauge potential [KSMP17], which may be obtained or approximated in different ways [CPSP19, HT21]. The additional counterdiabatic term aims to suppress non-adiabatic transitions that move the state away from the instantaneous ground state. Hence, approaches based on counterdiabatic terms can improve solution quality [CHP+22, CDS+24].

An extension of DCQO is bias-field DCQO (BF-DCQO), which is an iterative algorithm [CDS+24, RVC+24, isk]. In BF-DCQO, the information from the final evolved state is fed back as a bias for a new initial state, such that the initial Hamiltonian for another DCQO iteration is adapted accordingly. In addition to that, we consider a classical post-processing scheme where we perform a local search [DEH+23, DSE+24, WSW24] on the n_{keep} lowest energy bitstrings obtained from BF-DCQO at each iteration. This local search can be seen as a zero-temperature simulated annealing with n_s sweeps.

We conduct classical statevector simulations of quantum circuits from BF-DCQO for sequence lengths up to $n = 30$ using Qiskit 1.1.1, NumPy 2.0.2, and SciPy 1.14.1. We consider a maximum of 10 iterations, and n_{keep} was set to 50% of the measured states at each iteration. Furthermore, $n_s = 5$ sweeps were performed on each bitstring. In the context of these simulation results, we predict QPU execution times by making optimistic assumptions, i.e., that one shot on quantum hardware takes $2 \cdot 10^{-4}$ seconds, and the classical post-processing requires $\sim 10^{-7}$ seconds per sweep (for 30 qubits), based on the measured resources on an Apple M2 Pro chip. In Figure 17, we present the scaling of the time to measure the optimal solution for the classically simulated BF-DCQO combined with greedy classical post-processing for different sequence lengths. Additionally, we show the time-to-solution of QAOA with different numbers of layers p based on [SLC+24]. The time-to-solution is defined as $\tau \log(1-0.99)/\log(1-p_{gs})$ with τ the execution time per shot, p_{gs} is the success probability of finding the optimal solution, and the ratio $\log(1-0.99)/\log(1-p_{gs})$ accounts for the number of shots required to measure the optimal solution at least once with 99% probability. We emphasize that the reported runtimes for BF-DCQO were calculated based on early termination once the ground state was reached, rather than running all 10 iterations. In most cases, the ground state was obtained within 3 iterations; however, for cases beyond 28 qubits, up to 7 iterations were required. If we choose not to terminate after reaching the ground state, the success probability continues to be amplified by the bias field in successive iterations, contributing to a further reduction in the overall time-to-solution. Additionally, termination criteria can be defined by setting a threshold on the energy difference between successive iterations. Note that the BF-DCQO circuit requires as many entangling gates as QAOA ($p = 2$). In the studied range of system sizes, BF-DCQO scales better than QAOA ($p = 1$, $p = 4$) and is comparable to QAOA ($p = 8$, $p = 12$). Additionally, the scaling is also better with respect to exact classical solvers like Gurobi and CPLEX [SLC+24], as shown in Figure 18.

To test the performance experimentally, we selected the LABS problem with a sequence length of $n = 20$, which exceeds the largest LABS problem implemented on a quantum computer to date (i.e. $n = 18$ [SLC+24]), and present the results in the submission format introduced in Table 10 in Table 12.

Since the Hamiltonian corresponding to LABS is dense with both 2-local and 4-local terms, its realization is challenging due to the large number of entangling gates required. Therefore, we restrict the counterdiabatic terms to include only 2-local terms. We used the 156-qubit *ibm_marrakesh* device with `optimization level 2` transpilation, dynamical decoupling based on an XpXm sequence, and Qiskit 1.3.1. To facilitate the experimental implementation, we adopted the following strategy for initializing the bias fields. First, we created a random bitstring and applied a local greedy search with $n_s = 5$. Then, we use the resulting bitstring to compute the initial bias fields. We set the initial number of shots to $n_{\mathrm{shots}} = 500$ and $n_s = 2$. The n_{keep} parameter is chosen as 50% and n_{shots} is increased by 500 after each iteration. In Table 11, we present the results for ten different bias-field initializations. For the selected set of parameters, all the initializations resulted in a non-zero success probability. The $n = 20$ LABS problem has an eight-fold degenerate

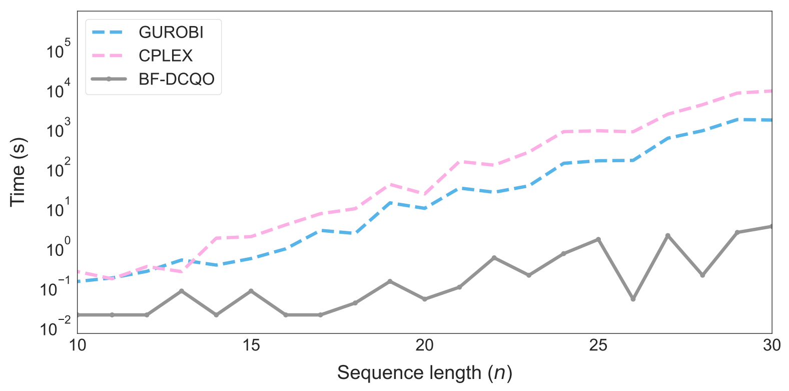


Figure 18: (LABS) Comparison of the time-to-solution for BF-DCQO, Gurobi, and CPLEX. The data for the classical optimizers was sourced from [SLC+24]. Note that the BF-DCQO times reflect idealized QPU runtime estimates, excluding transpilation.

Run	p_{gs}	Total Runtime (s)	Pre-Proc. (s)	QPU Runtime (s)	Post-Proc. (s)
1	0.10	32.23	0.213	32	0.01
2	0.23	32.25	0.241	32	0.01
3	0.37	32.23	0.216	32	0.01
4	0.45	32.24	0.226	32	0.01
5	0.10	32.24	0.225	32	0.01
6	0.25	96.68	0.665	96	0.01
7	0.49	64.44	0.427	64	0.01
8	0.13	64.42	0.408	64	0.01
9	0.08	128.87	0.854	128	0.01
10	0.12	96.64	0.630	96	0.01

Table 11: (LABS) Summary of results for ten experimental trials of BF-DCQO on the size $n = 20$ LABS problem executed on *ibm_marrakesh*. The overall runtime consists of pre-processing time, which includes circuit construction and transpilation, QPU runtime, which includes the payload circuit preparation and execution time, and post-processing time, which includes the local search post-processing. Further p_{gs} shows the probability of sampling the best solution–here the optimal one.

ground state, which can be divided into two distinct independent groups based on spin-flip and reflection symmetry. In our results, we measure optimal sequences from both groups. The average overall runtime to measure the optimal solution was 61.2 seconds. However, excluding the payload quantum circuit preparation, the average runtime becomes 9.1 seconds, which is still better than the two exact classical methods shown in Figure 18.

In summary, this section presents a benchmarking study of BF-DCQO against CPLEX, GUROBI, and QAOA for the LABS problem with sequence lengths of up to 30. Our empirical results demonstrate that BF-DCQO's runtime scaling to reach the optimal solution is comparable to that of QAOA with $p = 12$ layers, while requiring six times fewer entangling gates. Prior empirical studies have shown that QAOA with $p = 12$ outperforms the best-known classical approach, Memetic Tabu Search, by simulating sequence lengths of up to $n = 40$. Nevertheless, further research is needed to systematically assess whether BF-DCQO can ultimately achieve a definitive quantum speedup over state-of-the-art classical methods.

5.2 Minimum Birkhoff Decomposition

The following section presents a quantum variational algorithm for the Minimum Birkhoff Decomposition problem, which looks for a decomposition of an $n \times n$ doubly stochastic matrix with k permutation matrices, where k is as small as possible. The solutions found with state-of-the-art classical solvers are given in Section 4.3.4.

Problem Identifier	LABS ($n = 20$)
Submitter	Narendra N. Hegade (Kipu Quantum), Alejandro Gomez Cadavid (Kipu Quantum, UPV/EHU).
Date	January 20, 2025
Reference	[LAB]
Best Objective Value	26 (optimal)
Optimiality Bound	N/A
Modeling Approach	HUBO
#Decision Variables	20
#Binary Variables	20
#Integer Variables	0
#Continuous Variables	0
#Non-Zero Coefficients	90+525 (quadratic + quartic terms)
Coefficients Type	integer
Coefficients Range	{2, 4}
Workflow	Each iteration of the algorithm calls: 1) Sampling with BF-DCQO 2) Local search sweeps
Algorithm Type	Stochastic
#Runs	10
#Feasible Runs	10 (unconstrained problem)
#Successful Runs	10
Success Threshold	0 (requiring optimal solutions)
Hardware Specifications	CPU: Apple M2 Pro, QPU: *ibm_marrakesh*.
Total Runtime	61.22s
CPU Runtime	0.42s
GPU Runtime	N/A
QPU Runtime	60.80s
Other HW Runtime	N/A

Table 12: (LABS) Report of benchmark submission for the LABS instance with $n = 20$, also available in [QOBLIB]. All runtimes are reported as the average over the 10 runs.

The variational quantum algorithm considered here aims to solve the following problem

$$\underset{\theta\in[0,2\pi]^m}{\text{minimize}} \quad \underset{(P_1,\ldots,P_k)\sim\mathcal{P}(\theta)}{\mathbf{E}} \left[\left\| D - \sum_{i=1}^{k} c_i P_i \right\|_2 \right] \tag{19}$$

where $(P_1, \ldots, P_k)$ is *a collection* of k different permutation matrices drawn from a parameterized distribution $\mathcal{P}(\theta)$, and $(c_1, \ldots, c_k)$ are weights such that $\sum_{i=1}^{k} c_i = 1$. The distribution $\mathcal{P}(\theta)$ is prepared with a parameterized quantum circuit where $\theta := (\theta_1, \ldots, \theta_m) \in [0, 2\pi]^m$ controls m circuit gates. The variational algorithm runs for a sequence of iterations where θ is updated in each iteration to minimize the cost function $\mathbf{E}[\|D - \sum_{i=1}^{k} c_i P_i\|_2]$. The cost function is computed in two steps. First, we sample *a collection* of k different permutation matrices with a quantum circuit (*main permutation sampling*) and then compute the weights classically (*black-box optimization*).

To sample k permutation matrices, we use a quantum circuit with four layers. Each layer consists of RY(θ_j) gates ($j \in \{1, \ldots, m\}$) on every qubit and a CZ pairwise entanglement where qubit i is entangled with qubit $i+1$ for all even values of i, and then a second layer where qubit i is entangled with qubit $i+1$ for all odd values of i. We can encode k different permutation matrices as a bit string as follows. First, we write a permutation matrix as a sequence of integers using the Lehmer code [Leh60], and then map the sequence to a *unique* integer in $\{1, \ldots, n!\}$. That is, an integer $\{1, \ldots, n!\}$ corresponds to a permutation matrix. The next step is to map k permutation matrices (i.e., integers) to a unique integer using the combinatorial number system [SFT16]. The resulting encoding requires $\lceil \log_2 \binom{n!}{k} \rceil$ qubits, where $\binom{n!}{k}$ is the number of all possible solutions—i.e., all possible combinations of k permutations. For example, with $n = 4$ and $k = (n-1)^2 + 1$, there are a total of $\binom{4!}{10} = 1{,}961{,}256$ different permutation combinations and the encoding requires 21 qubits. Similarly, with $k = (n-1)^2+1$ and $n = 5$, there are $\binom{5!}{17} \approx 1.9 \cdot 10^{20}$ different permutation combinations and we require 68 qubits. The encoding process described can be reversed to map a bit string back to a unique collection of k permutation matrices. However, since $\log_2 \binom{n!}{k}$ may

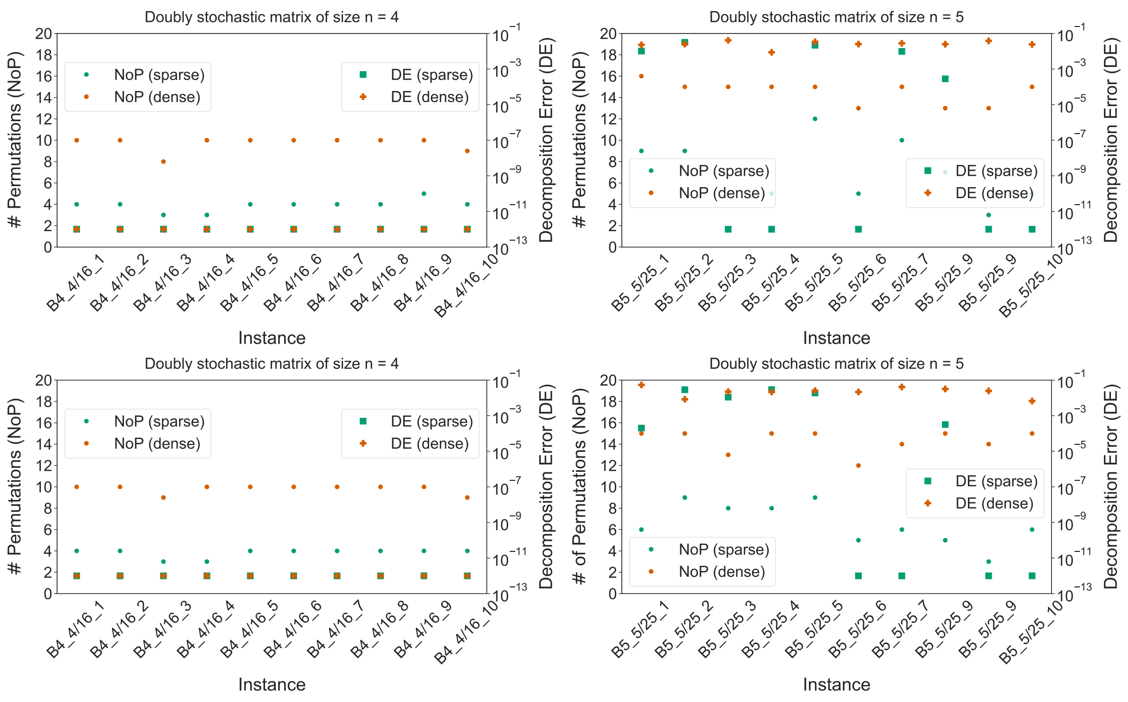


Figure 19: (Birkhoff) Quantum *simulation* (top) *hardware* (bottom) results for $n = 4$ (left) and $n = 5$ (right). The *error markers* with 10^{-12} error indicate an exact decomposition (i.e., zero error). The x-axis indicates the problem instance BX_sparse/dense_Z, where X is the size of the doubly stochastic matrix (i.e., n), sparse/dense is the number of permutations used to generate the doubly stochastic matrix (n for sparse and n^2 for dense), and Z is the instance number.

not be an integer, some bit strings may not correspond to permutation combinations. Thus, if the quantum circuit returns a non-valid bit string, we replace this with a valid bit string selected uniformly at random.

In the experiments presented below, we employed Optuna 4.1 [ASY+19] as a black-box solver for Eq. (19) to optimize the parameters θ. After sampling the permutation matrices, the coefficients $c_1, \dots, c_k$ are computed classically in two steps. In the first step (*black-box step 1*), we use CPLEX 22.1.1 [cpl25] to minimize $\|sD - \sum_{i=1}^{k} u_i P_i\|_2$ with respect to $u_i \in \{0, 1, 2, \dots, s\}$ subject to $0 \leq u_i \leq s$ and $\sum_{i=1}^{k} u_i = s$ where s is a given integer in the dataset that makes sD and interger-valued matrix. In the second step (*black-box step 2*), we use again CPLEX 22.1.1 to find the coefficients $c_i \in [0, 1]$ that provide the smallest decomposition subject to $\|D - \sum_{i=1}^{k} c_i P_i\|_2 \leq \|D - \frac{1}{s}\sum_{i=1}^{k} u_i P_i\|_2$ and $\sum_{i=1}^{k} c_i = 1$. Further details about the data encoding and the algorithmic flow of this variational quantum algorithm can be found in Appendix D.

We benchmark the variational algorithm using Qiskit 1.3.1 [JATK+24] with a simulator (*Qiskit Aer MPS*) and actual quantum hardware (*ibm_cusco*). All classical execution steps were run on a MacBook Pro (M1 Max CPU and 32 GB RAM). Each run of the algorithm consists of 10 iterations, where each iteration involves drawing 1024 shots from the quantum computer and hyperparameter optimization of the respective coefficients with a black-box solver, as explained above. We run the algorithm once for all instances with size $n = 4$ and $n = 5$, and present a performance summary in Figures 19. Recall from Section 4.3.3 that there are 10 instances per matrix size and density (sparse/dense) and that sparse/dense matrices are generated by sampling, respectively, n and n^2 permutation matrices and weights uniformly at random. Furthermore, Table 13 presents an example of a submission for a particular problem instance (B5_5_8) where we run the algorithm 10 times.

Figure 19 shows the simulation and hardware results for $n = 4$ and $n = 5$ with sparse and dense matrices; the dots indicate the number of permutations in the decomposition (left y-axis), and the squares/crosses represent the decomposition error (right y-axis). The x-axis indicates the problem instance BX_Y_Z, where X is the size of the doubly stochastic matrix (i.e., n), Y is the number of permutations used to generate the doubly stochastic matrix (n for sparse and n^2 for dense), and Z is the instance number. Observe from both figures that when $n = 4$ and the target matrix is sparse, the variational algorithm can consistently obtain sparse decompositions of length 3 and 4. For both simulation and hardware results, the solutions coincide with the solution obtained with CPLEX except for one instance (B4_4_9; simulation), where the decomposition length is 5 instead of 4 (CPLEX). Also, the decomposition is exact for all instances (i.e., zero error). With $n = 4$ and a dense target matrix, the performance remains similar to the sparse case. Finally, with $n = 5$, the performance degrades significantly compared to the $n = 4$ case, with only a few instances reaching sparse decompositions with decomposition errors below 10^{-10}. Despite solving only a few instances, the result is remarkable for a variational algorithm since there are $\binom{5!}{5} \sim 190$ million combinations of 5 permutation matrices of size 5. Moreover, some of the decompositions obtained with the variational algorithm are better than the decompositions obtained with approximate classical algorithms such as Birkhoff+ and Blended FW. For instance, with instance B5_5_6, Birkhoff+ obtains an exact decomposition of length 7, while the variational algorithm obtains an exact decomposition of length 5.

Problem Identifier	B5_5_8 ($n = 5$; density: sparse; instance number: 8)
Submitter	Mitsuharu Takeori (IBM Research – Tokyo)
	Víctor Valls (IBM Research Europe – Dublin)
Date	January 30, 2025
Reference	Appendix D
Best Objective Value	5
Optimality Bound	N/A
Modeling Approach	Quantum black-box optimization with CPLEX evaluated black-box
#Decision Variables	68 (main) + 17 (black-box step 1) + 34 (black-box step 2)
#Binary Variables	68 (main) + 17 (black-box step 2)
#Integer Variables	17 (black-box step 1)
#Continuous Variables	17 (black-box step 2)
#Non-Zero Coefficients	35 (black-box step 1) + 18 (black-box step 2)
Coefficients Type	integer
Coefficients Range	$[0, 10000]$ (black-box step 1)
Workflow	Each iteration of the variational algorithm calls:
	1) Main permutation sampling
	2) Black-box optimization step 1
	3) Black-box optimization step 2
Algorithm Type	Stochastic
#Runs	10
#Feasible Runs	10
#Successful Runs	4 (1st, 2nd, 5th, and 10th run)
Success Threshold	10^{-10}
Hardware Specifications	CPU: Apple M1 Max (Macbook Pro) , QPU: *ibm_cusco*.
Total Runtime	623.8s
CPU Runtime	603.7s
GPU Runtime	N/A
QPU Runtime	20.1s
Other HW Runtime	N/A

Table 13: (Birkhoff) Report of benchmark submission for instance B5_5_8 ($n = 5$; density: sparse; instance number: 8) from the Minimum Birkhoff Decomposition problem collection provided in [QOBLIB]. All runtimes are reported as the average over the 10 different runs.

5.3 Maximum Independent Set

Here, we benchmark standard QAOA [FGG14] on the independent set problem instances defined by graphs $G = (V, E)$ with $|V| = 17$ and $|V| = 52$ variables–see Section 4.7.3. Independent set is a constrained optimization problem which we can, thus, model as a QUBO by adding a penalty term

$$\max_{x} \sum_{i \in V} x_i - \lambda \sum_{(i,j) \in E} x_i x_j.$$

Here, λ is a positive Lagrange multiplier to introduce the constraints. We convert this QUBO to a Hamiltonian with the variable change $x_i = (1 - z_i)/2$ and promote the spin variables z_i to

Pauli-Z spin operators Z_i. The resulting Hamiltonian has two contributions: the objective and the constraints

$$H_{\text{obj}} = -\frac{1}{2}\sum_{i\in V} Z_i, \qquad \lambda H_{\text{const}} = -\frac{\lambda}{4}\sum_{(i,j)\in E} Z_iZ_j - Z_i - Z_j.$$

We neglect the irrelevant constant energy offsets, i.e., terms that are neither linear nor quadratic in Z_i. These terms are $|V|/2$, and $-\lambda|E|/4$ and stem from the objective and constraints, respectively. Therefore, the total cost function Hamiltonian is $H = H_{\text{obj}} + \lambda H_{\text{const}}$. To produce bitstrings from the quantum hardware, we sample from a QAOA trial wavefunction $|\psi(\beta,\gamma)\rangle = e^{-i\beta H_M}e^{-i\gamma H'}|+\rangle$. Here, $H_M = \sum_{i\in V} X_i$ is the standard QAOA mixer. H' is the cost operator $H_{\text{obj}} + \lambda H'_{\text{const}}$ used to design the trial wavefunction, and we apply $p = 1$ repetitions. Crucially, H'_{const} may implement only a subset of the constraints in H_{const} to limit the depth of the quantum circuit. We must now optimize the parameters β, γ, and λ. In a typical QAOA, this is done in a closed loop with the quantum hardware, which is time and resource-consuming. Best practices on how to carefully choose transpilation and error suppression strategies for a QAOA benchmark execution can be found on Github [Bes]. In our case, we choose to optimize β, γ, and λ on classical hardware by evaluating $\langle H\rangle$, which is efficiently possible for a depth-one ansatz [EMW21]. Optimizing λ is a hard task [ARCR+24]. Here, we chose to strike a balance between the impact of the actual objective part of the Hamiltonian and the constraint part of the Hamiltonian. This can be formulated, e.g., as the optimization problem

$$\max_{\lambda}\ \min\left(\langle H_{\text{obj}}\rangle^\star, \lambda\langle H_{\text{const}}\rangle^\star\right).$$

More specifically, we are aiming to find the largest λ, which still gives us the correct solution for the objective function underlying our problem. Here, the expectation values $\langle\cdot\rangle^\star$ are taken over $|\psi(\beta^\star,\gamma^\star)\rangle$ with classically evaluated optimal parameters $\beta^\star$ and $\gamma^\star$ that maximize $\langle H\rangle$ at a fixed λ. Notably, $\beta^\star$ and $\gamma^\star$ are found from the inner optimization

$$\max_{\beta,\gamma}\ \langle\psi(\beta,\gamma)|H_{\text{obj}} + \lambda H_{\text{const}}|\psi(\beta,\gamma)\rangle = \max_{\beta,\gamma}\ E(\beta,\gamma;\lambda).$$

It should be noted that the operator in the expectation value is now the full cost Hamiltonian H and not the potentially simplified H'. The parameters β and γ are optimized with COBYLA, provided by ScipPy. Crucially, these QAOA parameters depend on the Lagrange multiplier λ. Therefore, the Lagrange multiplier optimization is expressed as

$$\max_{\lambda}\ \min\left(\langle H_{\text{obj}}\rangle^\star, \langle H\rangle^\star - \langle H_{\text{obj}}\rangle^\star\right) = \max_{\lambda} F(\lambda).$$

We optimize λ by performing a linear scan over the interval [0, 1] and [0, 0.2] for the 17 and 52 qubit instances, respectively, followed by a refinement using COBYLA from SciPy [VGO+20]. The resulting optimized λ values are 0.528 and 0.119 for the 17 and 52 qubit instances, respectively.

n	Total Runtime	Pre-Proc. (s) ($t_{\text{opt.}}; t_{\text{circ}}$)	QPU Runtime (s)	Post-Proc. (s)	Best Obj.Val.
17	77	36 (22; 14)	41	0	4 (optimal)
52	227	187 (175;12)	38	2	13 (optimal)

Table 14: (Independent Set) Summary of benchmarking results for mamila-kangaroo and aves-sparrow stable set instances. The process was executed once. The Overall Runtime, reported in seconds, is broken down into (i) the pre-processing time which includes the time t_{opt} to obtain $\beta^\star$, $\gamma^\star$, $\lambda^\star$ and t_{circ} to create the quantum circuits, (ii) the QPU runtime which includes the payload quantum circuit preparation and execution (queuing time is not included), and (iii) the time taken to post-process the samples.

We would require $\lambda > 1$ to make the lowest energy state of H correspond to the optimal solution of the MIS problem. However, we can only afford large values of lambda if we run deep QAOAs with $p > 1$ since they have a better resolution than QAOAs with $p = 1$. Since hardware noise restricts us to shallow circuits, we use $p = 1$ and work with small values of λ. Once the optimal parameters are found, we create a quantum circuit corresponding to $|\psi(\beta^\star(\lambda^\star),\gamma^\star(\lambda^\star);\lambda^\star)\rangle$ and sample 1024 candidate solutions on *ibm_fez*, a 156 qubit Heron R2 superconducting qubit processor [IBM]. These circuits are executed in *session mode* as this allows us to report the QPU time as the payload quantum circuit preparation and execution time–as discussed in Section 3.3.2. Finally, the samples are post-processed by the greedy classical method described in Appendix E. This benchmark was executed using Numpy 1.23.5, Qiskit 1.2.2, and SciPy 1.11.3, all running on Python 3.11.5. We benchmark this approach on stable set instances referred to as *mamila kangoroo* and *aves sparrow* with 17 and 52 variables, which are given in [QOBLIB]. All constraints of the 17-variable instance are implemented on hardware, i.e., $H'_{\text{const}} = H_{\text{const}}$. This requires a total of 308 CNOT gates. For the 52-variable instance, we only implement 16.7% of the constraints by allowing three layers of SWAP gates [WVG+22]. We, therefore, only implement the constraints that can be built up on a line of qubits on which we apply three layers of SWAP gates as described in Ref. [WVG+22]. The resulting circuit has a total of 207 CNOT gates. This keeps the fidelity of the circuit, approximated by $(1 - E_{\text{CZ}})^{n_{\text{CZ}}}$, above 50%. Here, E_{CZ} and n_{CZ} are the median error per two-qubit CZ gate and the number of two-qubit CZ gates, respectively. With the 52-variable instance, we do not sample any feasible solutions. However, after the greedy classical post-processing, we obtained the optimal sample. The runtime results from the process described above are reported in Table 14. As expected, most of the runtime is dedicated to optimizing the parameters β, γ, and λ. We observe that 60% and 27% of the post-processed samples for the 17 and 52 decision variable problems, respectively, are optimal, see Figure 20. Finally, Table 15 presents an example of a submission report for the 52−variable instance.

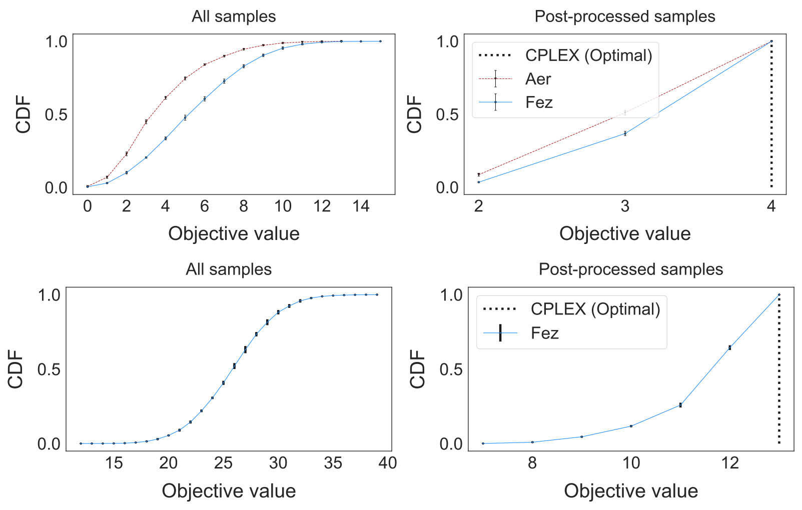


Figure 20: (Independent Set) Cumulative distribution of the objective value corresponding to the samples for the mamila kangaroo (top) and the aves sparrow (bottom) problem instances before classical post-processing (left) and after post-processing of the quantum samples (right). The mamila kangaroo problem is small enough to simulate the QAOA on a classical processor (dashed-dotted line). The samples from the *ibm_fez* quantum processor are shown as the solid blue line.

6 Conclusion and Discussion

It is widely believed that quantum computers can achieve an advantage over classical computers for *some* optimization problem instances in the future. While complexity theory suggests that, for **NP**-hard problems, the speed-up for finding provably optimal solutions is in general at most quadratic [BBBV97, dW23, DH99, GWG21], better quantum approximation algorithms with an exponential speed-up [JSW+24] or quantum optimization heuristics that outperform classical algorithms on certain instances may still exist. However, practical demonstrations of quantum advantage in optimization are still missing.

To identify problems where quantum advantage might be possible and to track progress toward achieving it, systematic analysis and benchmarking are key. To this end, we introduce [QOBLIB], an open-source quantum optimization benchmarking library. We present ten classes of combinatorial optimization problems with instances of varying size and complexity for benchmarking classical and quantum optimization algorithms. Further, we define a reporting standard that ensures reproducibility and fair comparison while remaining flexible for evaluating different algorithms and hardware platforms.

Most problems are challenging for state-of-the-art classical solvers even at moderate size, and many of them are related to use cases of practical relevance. The problem classes exhibit diverse properties, such as different densities and types of constraints, to enable the study of different

Problem Identifier	aves-sparrow-social.gph
Submitter	Daniel J. Egger (IBM Research Europe – Zurich)
Date	January 15th, 2025
Reference	[QAO]
Best Objective Value	13
Optimality Bound	N/A
Modeling Approach	QUBO
#Decision Variables	52
#Binary Variables	52
#Integer Variables	0
#Continuous Variables	0
#Non-Zero Coefficients	506 (the number of edges in the graph + number of nodes)
Coefficients Type	continuous
Coefficients Range	$[0, 2]$
Workflow	1) The parameters β and γ of the depth-one QAOA and the Lagrange multipliers are optimized classically. 2) Samples are drawn from the QPU. 3) Samples from the QPU are classically post-processed.
Algorithm Type	Stochastic
#Runs	5
#Feasible Runs	5
#Successful Runs	5
Success Threshold	0
Hardware Specifications	CPU: Intel Core i9-10885H (Levono P1), QPU: *ibm_fez*.
Total Runtime	252s
CPU Runtime	192s
GPU Runtime	N/A
QPU Runtime	60s
Other HW Runtime	N/A

Table 15: (Independent Set) Benchmark submission, for instance aves-sparrow-social from the independent set benchmarking problem collection provided in the [QOBLIB] repository.

mathematical formulations and algorithmic approaches. This is essential for identifying problem structures suitable for quantum advantage in optimization.

We provide classical baseline results for all problem classes and quantum baseline results for selected ones to kick off this effort. However, performing exhaustive benchmarks is a massive endeavor and can only be achieved by the broader scientific community. Thus, we invite all researchers interested in advancing (quantum) optimization to investigate the provided problem instances and submit solutions using algorithms and hardware of their choice. We also want to emphasize the importance of improving classical methodologies, amongst other reasons, because quantum advantage can only be claimed if the best-known classical methods are outperformed. To accelerate the overall research, we also encourage researchers to report results for approaches that did not work well. To conclude, [QOBLIB] is intended to be a living repository: novel solutions will be included on a rolling basis, and novel problem instances or adaptations of existing ones may be added as the state-of-the-art evolves.

Acknowledgments

We want to thank Karen Aardal at TU-Delft for providing results for the Market Split instances using lattice point enumeration and Nils-Christian Kempke at ZIB for providing results by GPU-accelerated Schroeppel Shamir enumeration. We would like to thank Sébastien Designolle at the Zuse Institute Berlin for useful discussions on the Blended FW algorithm used in the classical benchmarking of the minimum Birkhoff decomposition problem (Section 4.3.4). We also thank Julien Gacon for valuable discussions and a helpful review, and Takashi Imamichi for assistance with software issues, sharing his insights, and providing feedback. Additionally, we are grateful to Andrew Wack and Paul Nation for insightful conversations about the benchmarking of quantum algorithms and their feedback on this work.

Part of the work for this article has been conducted in the Research Campus MODAL funded by the German Federal Ministry of Education and Research (BMBF) (fund numbers 05M14ZAM, 05M20ZBM, 05M2025). GN is supported by ONR award # N000142312585. YC would like to acknowledge the support from the Quantum Engineering Program project A-8000339-01-00, from the Ministry of Education Singapore under project A-8000014-00-00, and by the National Research Foundation Singapore under its Industry Alignment Fund – Pre-positioning (IAF-PP) Funding Initiative and the Monetary Authority of Singapore under A-0003504-17-00. DVB would like to acknowledge the support of the Research Foundation Flanders (FWO) [23AXE35N].

Author Contributions

This review was written as part of the Quantum Optimization Working Group initiated in July 2023 by IBM Quantum and its partners.
TKo: Conceptualization, Software, Investigation, Resources, Data Curation, Writing–Original Draft, Writing–Review and Editing, Supervision; DEBN: Conceptualization, Writing–Original Draft, Writing–Review and Editing; YC: Conceptualization, Investigation, Modeling, Resources, Writing–Original Draft; GC: Conceptualization, Writing–Review and Editing; DJE: Investigation, Data Curation, Writing–Original Draft; RHe: Investigation, Resources, Writing–Original Draft; NNH: Investigation, Resources, Writing–Original Draft; AGC: Investigation, Resources, Writing–Original Draft; RHu: Software, Data Curation; TIm: Investigation, Data Curation; TIt: Writing–Original Draft; TKl: Resources, Writing–Original Draft; PMX: Writing–Original Draft, Investigation, Software; NM: Investigation, Data Curation, Writing–Original Draft; JAMB: Writing - Review and Editing ; KN: Software, Investigation; GN: Conceptualization, Methodology, Software, Investigation, Resources, Data Curation, Writing–Original Draft, Writing–Review and Editing; COM: Conceptualization, Writing-Review and Editing; JP: Investigation, Resources; MP: Writing–Original Draft; AR: Writing–Original Draft, Investigation; MS: Software, Investigation, Resources, Data Curation; NS: Software, Investigation; MT: Investigation; VV: Methodology, Software, Investigation, Resources, Data Curation, Writing–Original Draft; DVB: Software, Investigation, Data Curation; SW: Conceptualization, Writing–Original Draft, Writing–Review and Editing, Supervision; CZ: Conceptualization, Writing–Original Draft, Writing–Review and Editing, Supervision

A Building Capabilities via Instance Simplification

The problem instances we propose are challenging to even implement on current quantum computers due to noise, which limits the circuit depth. To track the progress of quantum algorithms and hardware today we therefore need a spectrum of simplified problem instances to observe a quantum signal. These instances will often be easy to solve classically. We propose to generate simplified QUBOs such that the time-evolution of the corresponding Hamiltonian becomes less difficult. More specifically, we consider a line of n qubits which achieves all-to-all connectivity with a SWAP network with $n-2$ layers of SWAP gates [WVG+22]. Crucially, each entry (i, j) of the QUBO matrix can be implemented after $d(i, j) \in \{0, ..., n-2\}$ layers of SWAP gates. The distances $d_{i,j}$ allow us to define an $n \times n$ mask where entry $m_{i,j}(k)$ is one if $d(i, j) \leq k$ and zero otherwise. The number of allowed layers k thus controls the difficulty of implementing $\exp(-iH_C)$ on hardware. We simplify a QUBO matrix with this mask through the mapping $Q_{i,j} \to Q'_{i,j} = m_{i,j}(k)Q_{i,j}$. Crucially, this only makes sense if the resulting QUBO Q' remains connected. In the benchmarking repository, we provide a code to simplify a QUBO matrix following this procedure. Crucially, in this process, the code generates a named field which contains the number of SWAP layers that constrain the QUBO for traceability.

B Market Split Classical Results

Fig 21 shows the results for Gurobi on a 32-core AMD EPYC-7542 processor using 8 cores and ABS2 [NTI+23] on an A100 SXM4 80GB, respectively, for different sizes of instances with different parameters m. ABS2 is very fast for smaller instances, but fails to find solutions within one hour for instances larger than size $m = 6$.

C Market Split Instance Generation

To benchmark solvers, it is essential to generate feasible instances of this problem. A naive approach is to randomly sample the entries of A from a uniform distribution and compute b accordingly. However, this approach often results in infeasible instances, which are less useful for benchmarking. For instances that were too large to be tested, i.e., $m \geq 7$, we constructed instances as follows:

1. Generate a Random Solution Vector: Create a random solution vector x where approximately half (± 2) of the entries are set to 1, and the rest are set to 0.

2. Generate Matrix A: Sample the entries of A uniformly from a specified range D.

3. Compute the Right-Hand Side b: Compute each entry of b as:

$$b_i = \left\lfloor \frac{\sum_{j\in[n]} a_{ij}}{2} \right\rfloor$$

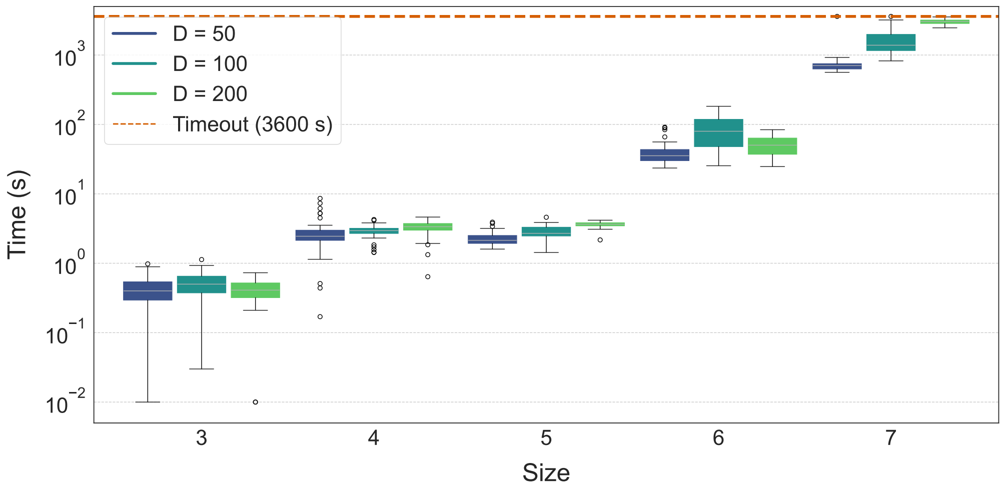
4. Adjust for Feasibility: Ensure the feasibility of the generated instance using the following steps for each row:

   - Step 1: Repeat until no more improvement is possible:
     Identify a pair of entries in the row where one corresponds to 0 in x and the other to 1. Perform a “switch” that reduces the absolute slack of the row as much as possible.
   - Step 2: If the row is still not feasible:
     Calculate the slack s for the row. Set $c = \frac{3 \cdot \text{numones}}{2}$, where num_ones is the number of 1’s in the row. Find c pairs of 0 and 1, and adjust the corresponding entries by adding $\frac{s}{c}$ to the 1’s and subtracting it from the 0’s.
   - Step 3: Repeat this adjustment until the row becomes feasible.

   In most cases, only a few switches and adjustments are sufficient to achieve feasibility.

D Minimum Birkhoff Decomposition Quantum Baseline Benchmark – Additional Details

This section presents additional details that facilitate the reproducibility of the quantum baseline benchmarking results for the minimum Birkhoff decomposition problem presented in Section 5.2.



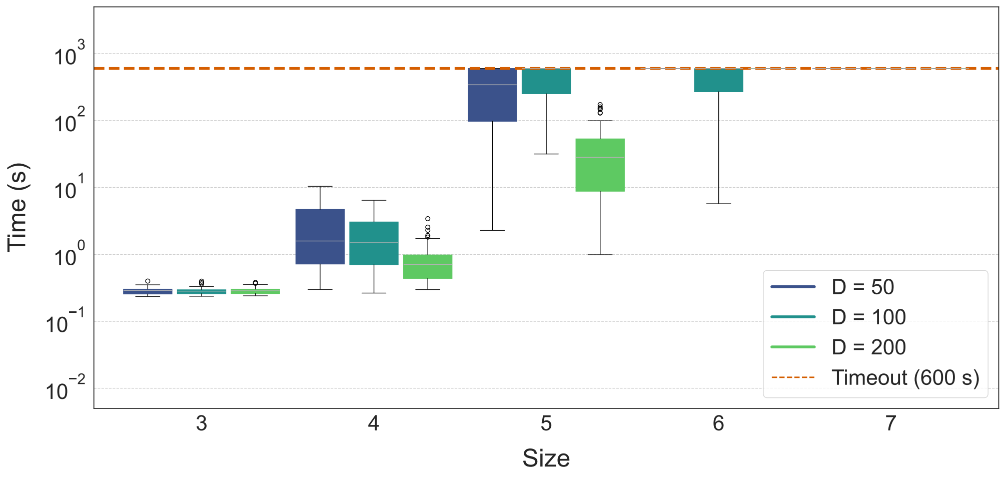


Figure 21: Distribution of solution times using (top) Gurobi 11.0.0 with a time limit of 3600s and (bottom) ABS2 with a time limit of 600s for instances with different sizes m and coefficients range D. All trials for $m \geq 8$ timed out for Gurobi and for $m \geq 7$ timed out for ABS2.

Model Encoding

The following example illustrates the encoding strategy used in the variational algorithm, which was employed to generate the results presented in Section 5.2.

Permutation to bistring. Consider the following 4×4 permutation matrix

$$\begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Recall there are $4! = 24$ permutation matrices. We can write the permutation matrix as an array

$$p = [2, 3, 1, 4].$$

The Lehmer code [Leh60] maps p to another array l where the i-th entry of l is equal to $l_i = \sum_{j=1}^{i-1} 1_{p_j < p_i}$. That is, l_i is equal to the number of values $p_1, \ldots, p_{i-1}$ smaller than p_i. For the permutation array $p = [2, 3, 1, 4]$, the Lehmer code mapping is

$$l = [0, 1, 0, 3].$$

Next, we can map the Lehmer code to an integer in $[0, 4! - 1]$. In our example, the mapping of $[0, 1, 0, 3]$ to an integer is as follows:

$$0! \cdot 0 + 1! \cdot 1 + 2! \cdot 0 + 3! \cdot 3 = 19,$$

where 19 corresponds to the bit string 10011.

Bitstring to Permutation. Consider the bitstring 10011. We can map the bit string to the integer 19, and 19 to the array $[0, 1, 0, 3]$ by reversing the process used above. Similarly, we can map $[0, 1, 0, 3]$ to $p = [2, 3, 1, 4]$ and so recover the permutation matrix. Finally, note that while we can always map a permutation matrix to a bit string, the inverse is not always possible. An example is the bit string 11111 (31) since there are 24 permutation matrices, and the encoding above maps a permutation matrix to an integer in $\{0, 1, 2, \dots, 4! - 1\}$. If such a case occurs, we return a random bit string in the feasible range $\{0, 1, 2, \dots, n! - 1\}$.

Mapping k unique integers to an integer. Suppose we have $k = 4$ different integers $[17, 19, 18, 16]$, each representing a unique permutation matrix. Next, we order the integers in ascending order:

$$a = [16, 17, 18, 19].$$

Map the integers to a unique integer using the combinatorial number system, i.e., $b = \sum_{i=1}^{k} \binom{a_i}{i}$. In our example,

$$\binom{16}{1} + \binom{17}{2} + \binom{18}{3} + \binom{19}{4} = 4844.$$

Mapping an integer to k unique integers. Suppose we want to map 4844 to $k = 4$ unique integers. We start by finding the largest integer x such that $\binom{x}{4} \leq 4844$. In this case, x is equal to 19. Repeat the process for $4844 - \binom{19}{4} = 968$ and $k - 1$, i.e., find x such that $\binom{x}{3} \leq 968$, which is 18. Repeating the process until the remainder is zero gives the integers $[19, 18, 17, 16]$.

Ansatz and black-box optimization

The variational part of the algorithm samples bit strings using the following variational circuit (Qiskit 1.3.1):

```
ansatz = n_local(
    num_qubits = n,
    rotation_blocks = "ry",
    entanglement_blocks = 'cz',
    entanglement = 'pairwise',
    reps = 4
    )
```

Furthermore, the optimization of the parameters is carried out with Optuna 4.1 [ASY+19] with the CMAES sampler. There are a total of 10 optimization rounds/iterations, where each round measures 1024 shots with the quantum hardware or simulator, respectively. For each shot, the black-box computes the objective value as described in Section 5.2. Recall that the black-box optimization takes as an input a collection of k permutation matrices, the target matrix D, and an integer s such that sD is integer-valued. The following is a Python (3.10) implementation of the black-box optimization using CPLEX (22.1.1):

```
 ########## Black-box optimization input  ##########
A # $n^2 \times k$ matrix where columns are permutation matrices
b # Target permutation matrix as column vector
scale # integer that makes A * scale an integer matrix
k # maximum decomposition length

########## Black-box optimization: STEP 1 ##########

model = Model()
w = model.integer_var_list(A.shape[1])

# add constraints
model.add_constraint(
    model.sum(w[index] for index in range(0,A.shape[1])) == scale)
for i in range(0,A.shape[1]):
    model.add_constraint(w[i] >= 0)
for i in range(0,A.shape[1]):
    model.add_constraint(w[i] <= scale)
```

```
# add objective
model.minimize(model.sum_squares(
    (model.sum(A[i,j]*w[j] for j in range(0,A.shape[1])) - b[i])
            for i in range(0,A.shape[0])))

# solve
solution = model.solve()
sol = solution.get_value_list(w)

# compute error  for step 2
dense_val = np.linalg.norm(A @ sol - b,2)

########## Black-box optimization: STEP 2 ##########
model = Model()
w = model.continuous_var_list(k)
y = model.binary_var_list(k)
p = np.asarray((P.flatten()))[0]

# add constraints
model.add_constraint(model.sum(w[index]
    for index in range(0,k)) == 1)
for i in range(0,k):
    model.add_constraint(w[i] >= 0)
for i in range(0,k):
    model.add_constraint(w[i] <= 1)
for i in range(0,k):
    model.add_constraint(w[i] <= y[i])

# add objective
model.add_constraint((model.sum_squares(
    (model.sum(w[j]*p[i + j*n*n] for j in range(0,k)) - b[i])
            for i in range(0,n*n))) <= dense_val)
model.minimize(model.sum(y[i] for i in range(0,k)))

# solve
solution = model.solve()
```

Solution for the submission in Table 13

The doubly stochastic matrix in instance B5_5_8 is:

$$D = \begin{bmatrix} 0.0143 & 0.6351 & 0.0488 & 0.151 & 0.1508 \\ 0.7859 & 0.1998 & 0 & 0.0143 & 0 \\ 0 & 0 & 0.151 & 0.1508 & 0.6982 \\ 0.151 & 0.1508 & 0.6494 & 0.0488 & 0 \\ 0.0488 & 0.0143 & 0.1508 & 0.6351 & 0.151 \end{bmatrix}.$$

The decomposition obtained in the four successful runs is:

$$D = 0.0143 \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix} + 0.6351 \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix} + 0.1508 \begin{bmatrix} 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$
$$+ 0.151 \begin{bmatrix} 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} + 0.0488 \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

The decomposition length obtained with the variational quantum algorithm coincides with the decomposition length found with the integer program formulation solved with CPLEX (see formulation in Section 4.3.4).

E Maximum Independent Set Classical Post-processing

Here, to keep the paper self-contained, we provide the classical algorithm with which the samples drawn from the quantum computer are post-processed in the maximum independent set benchmark. Algorithm 1 first makes a given sample x feasible by removing nodes from the independent set until all the constraints are satisfied. Second, Algorithm 1 adds nodes back into the independent set as long as they do not introduce any violations of the constraints.

Algorithm 1 Greedy post-processing of candidate solutions x to make x feasible, i.e., an independent set, and enlarge the set while remaining feasible.

```
Require: candidate solution $x \in \{0,1\}^n$
  $v \leftarrow$ compute_violations($x$)      ▷ map vertex to the number of constraints it violates
  $c \leftarrow V$                             ▷ $c$ are vertices we try to add to the stable set

  while sum($v$.values()) $> 0$ do             ▷ Make $x$ feasible
      $v_{\max} \leftarrow$ vertex that violates the largets number of constraints
      $x[v_{\max}] \leftarrow 0$
      $c$.remove($v_{\max}$)
      $v \leftarrow$ compute_violations($x$)
  end while

  while len($c$) $> 0$ do                      ▷ Add vertices to the stable set and remain feasible
      $u \leftarrow c$.pop()
      $x' \leftarrow x$
      $x'[u] \leftarrow 1$
      if is_feasible($x'$) then
          $x \leftarrow x'$
      end if
  end while
  return $x$
```

References

[AAA$^+$24] Amira Abbas, Andris Ambainis, Brandon Augustino, Andreas Bärtschi, Harry Buhrman, Carleton Coffrin, Giorgio Cortiana, Vedran Dunjko, Daniel J. Egger, Bruce G. Elmegreen, Nicola Franco, Filippo Fratini, Bryce Fuller, Julien Gacon, Constantin Gonciulea, Sander Gribling, Swati Gupta, Stuart Hadfield, Raoul Heese, Gerhard Kircher, Thomas Kleinert, Thorsten Koch, Georgios Korpas, Steve Lenk, Jakub Marecek, Vanio Markov, Guglielmo Mazzola, Stefano Mensa, Naeimeh Mohseni, Giacomo Nannicini, Corey O'Meara, Elena Peña Tapia, Sebastian Pokutta, Manuel Proissl, Patrick Rebentrost, Emre Sahin, Benjamin C. B. Symons, Sabine Tornow, Víctor Valls, Stefan Woerner, Mira L. Wolf-Bauwens, Jon Yard, Sheir Yarkoni, Dirk Zechiel, Sergiy Zhuk, and Christa Zoufal. Challenges and opportunities in quantum optimization. *Nature Reviews Physics*, October 2024.

[AAPS24] Arturo Acuaviva, David Aguirre, Rubén Peña, and Mikel Sanz. Benchmarking quantum computers: Towards a standard performance evaluation approach, 2024.

[ABCC03] David Applegate, Robert Bixby, Vašek Chvátal, and William Cook. Implementing the Dantzig-Fulkerson-Johnson Algorithm for Large Traveling Salesman Problems. *Mathematical programming*, 97:91–153, 2003.

[ABCG17] Martin Anthony, Endre Boros, Yves Crama, and Aritanan Gruber. Quadratic reformulations of nonlinear binary optimization problems. *Math. Program.*, 162(1–2):115–144, March 2017.

[ABH$^+$00] Karen Aardal, Robert E. Bixby, Cor A. J. Hurkens, Arjen K. Lenstra, and Job W. Smeltink. Market split and basis reduction: Towards a solution of the cornuéjols-dawande instances. *INFORMS Journal on Computing*, pages 192–202, 2000.

[ABW02] Andrew P Armacost, Cynthia Barnhart, and Keith A Ware. Composite variable formulations for express shipment service network design. *Transportation science*, 36(1):1–20, 2002.

[AKT08] Tobias Achterberg, Thorsten Koch, and Andreas Tuchscherer. On the effects of minor changes in model formulations. Technical Report 08-29, ZIB, Takustr. 7, 14195 Berlin, 2008.

[ARCR$^+$24] Edoardo Alessandroni, Sergi Ramos-Calderer, Ingo Roth, Emiliano Traversi, and Leandro Aolita. Alleviating the quantum Big-M problem, 2024.

[Aro96] Sanjeev Arora. Polynomial time approximation schemes for euclidean tsp and other geometric problems. In *Proceedings of 37th Conference on Foundations of Computer Science*, pages 2–11. IEEE, 1996.

[ASY$^+$19] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 2623–2631, 2019.

[AVKS98] Pankaj K Agarwal, Marc Van Kreveld, and Subhash Suri. Label placement by maximum independent set in rectangles. *Computational Geometry*, 11(3-4):209–218, 1998.

[AZ10] Doron Avramov and Guofu Zhou. Bayesian Portfolio Analysis. *Annual Review of Financial Economics*, 2(1):25–47, 2010.

[BBBV97] Charles H Bennett, Ethan Bernstein, Gilles Brassard, and Umesh Vazirani. Strengths and weaknesses of quantum computing. *SIAM journal on Computing*, 26(5):1510–1523, 1997.

[BBD$^+$22] Sebastian Brandhofer, Daniel Braun, Vanessa Dehn, Gerhard Hellstern, Matthias Hüls, Yanjun Ji, Ilia Polian, Amandeep Singh Bhatia, and Thomas Wellens. Benchmarking the performance of portfolio optimization with qaoa. *Quantum Information Processing*, 22(1):25, 2022.

[BBDvdB$^+$20] Thomas Bartz-Beielstein, Carola Doerr, Daan van den Berg, Jakob Bossek, Sowmya Chandrasekaran, Tome Eftimov, Andreas Fischbach, Pascal Kerschke, William La Cava, Manuel Lopez-Ibanez, Katherine M. Malan, Jason H. Moore, Boris Naujoks, Patryk Orzechowski, Vanessa Volz, Markus Wagner, and Thomas Weise. Benchmarking in Optimization: Best Practice and Open Issues, 2020.

[BBS$^+$24] David E Bernal Neira, Robin Brown, Pratik Sathe, Filip Wudarski, Marco Pavone, Eleanor G Rieffel, and Davide Venturelli. Benchmarking the operation of quantum heuristics and ising machines: scoring parameter setting strategies on optimization applications. *arXiv preprint arXiv:2402.10255*, 2024.

[BCGT98] Daniel Bienstock, Sunil Chopra, Oktay Günlük, and Chih-Yang Tsai. Minimum cost capacity installation for multicommodity network flows. *Mathematical programming*, 81:177–199, 1998.

[BDS10] D. Briskorn, A. Drexl, and F. C. R. Spieksma. Round robin tournaments and three index assignments. *4OR-Q J Oper Res*, 8:365–374, 2010.

[BEP05] Cristina Bazgan, Bruno Escoffier, and Vangelis Th. Paschos. Completeness in standard and differential approximation classes: Poly-(D)APX- and (D)PTAS-completeness. *Theoretical Computer Science*, 339(2-3):272–292, 2005.

[Ber13] Timo Berthold. Measuring the impact of primal heuristics. *Operations Research Letters*, 41(6):611–614, 2013.

[Bes] Best practices in quantum optimization . https://github.com/qiskit-community/qopt-best-practices. [Online; accessed April-3-2025].

[BG95] Daniel Bienstock and Oktay Günlük. Computational experience with a difficult mixedinteger multicommodity flow problem. *Mathematical programming*, 68:213–237, 1995.

[BGC$^+$23] David Van Bulck, Dries Goossens, Jan-Patrick Clarner, Angelos Dimitsas, George H. G. Fonseca, Carlos Lamas-Fernandez, Martin Mariusz Lester, Jaap Pedersen, Antony E. Phillips, and Roberto Maria Rosati. Which algorithm to select in sports timetabling?, 2023.

[BH14] Maciej Besta and Torsten Hoefler. Slim fly: A cost effective low-diameter network topology. In *SC'14: proceedings of the international conference for high performance computing, networking, storage and analysis*, pages 348–359. IEEE, 2014.

[BHP$^+$25] Federico Berto, Chuanbo Hua, Junyoung Park, Laurin Luttmann, Yining Ma, Fanchen Bu, Jiarui Wang, Haoran Ye, Minsu Kim, Sanghyeok Choi, Nayeli Gast Zepeda, André Hottung, Jianan Zhou, Jieyi Bi, Yu Hu, Fei Liu, Hyeonah Kim, Jiwoo Son, Haeyeon Kim, Davide Angioni, Wouter Kool, Zhiguang Cao, Qingfu Zhang, Joungho Kim, Jie Zhang, Kijung Shin, Cathy Wu, Sungsoo Ahn, Guojie Song, Changhyun Kwon, Kevin Tierney, Lin Xie, and Jinkyoo Park. Rl4co: an extensive reinforcement learning for combinatorial optimization benchmark, 2025.

[Bie96] Daniel Bienstock. Computational study of a family of mixed-integer quadratic programming problems. *Mathematical Programming*, 74:121–140, 1996.

[Bir] Code for Birkhoff Decomposition . https://github.com/vvalls/BirkhoffDecomposition.jl. [Online; accessed April-3-2025].

[Bir46] Garrett Birkhoff. Tres observaciones sobre el algebra lineal. *Univ. Nac. Tucuman, Ser. A*, 5:147–154, 1946.

[BKS21] T. Berthold, T. Koch, and Y. Shinano. MILP. Try. Repeat. In P. De Causmaecker, E. Özcan, and G. Vanden Berghe, editors, *Proc. 13th Int. Conf. Pract. Theory Autom. Timetabling*, volume 2, pages 403–411. PATAT, 2021.

[BL92] Fischer Black and Robert Litterman. Global Portfolio Optimization. *Financial Analysts Journal*, 48(5):28–43, 1992.

[BMZ$^+$24] Jieyi Bi, Yining Ma, Jianan Zhou, Wen Song, Zhiguang Cao, Yaoxin Wu, and Jie Zhang. Learning to handle complex constraints for vehicle routing problems, 2024.

[Boe67] A Boehmer. Binary pulse compression codes. *IEEE Transactions on Information Theory*, 13(2):156–167, 1967.

[BPTW19] Gábor Braun, Sebastian Pokutta, Dan Tu, and Stephen Wright. Blended conditional gradients: the unconditioning of conditional gradients, 2019.

[Bru82a] Richard A Brualdi. Notes on the birkhoff algorithm for doubly stochastic matrices. *Canadian Mathematical Bulletin*, 25(2):191–199, 1982.

[Bru82b] Richard A. Brualdi. Notes on the birkhoff algorithm for doubly stochastic matrices. *Canadian Mathematical Bulletin*, 25(2):191–199, 1982.

[BS96] Cynthia Barnhart and Rina R Schneur. Air network design for express shipment service. *Operations Research*, 44(6):852–863, 1996.

[BS11] David B Brown and James E Smith. Dynamic portfolio optimization with transaction costs: Heuristics and dual bounds. *Manage. Sci.*, 57(10):1752–1770, October 2011.

[BSS10] David B Brown, James E Smith, and Peng Sun. Information relaxations and duality in stochastic dynamic programs. *Oper. Res.*, 58(4-part-1):785–801, August 2010.

[BTN99] A. Ben-Tal and A. Nemirovski. Robust solutions of uncertain linear programs. *Operations Research Letters*, 25(1):1–13, 1999.

[BVAV16] Shaileshh Bojja Venkatakrishnan, Mohammad Alizadeh, and Pramod Viswanath. Costly circuits, submodular schedules and approximate carathéodory theorems. In *Proceedings of the 2016 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Science*, pages 75–88, 2016.

[CCH00] Cheng-Shang Chang, Wen-Jyh Chen, and Hsiang-Yi Huang. Birkhoff-von neumann input buffered crossbar switches. In *Proceedings IEEE INFOCOM 2000. Conference on Computer Communications. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies (Cat. No.00CH37064)*, volume 3, pages 1614–1623 vol.3, 2000.

[CCJ90] Brent N. Clark, Charles J. Colbourn, and David S. Johnson. Unit disk graphs. *Discrete Mathematics*, 86(1):165–177, 1990.

[CCMS74] Vinton G Cerf, Donald D Cowan, Ronald C Mullin, and RG Stanton. A lower bound on the average shortest path length in regular graphs. *Networks*, 4(4):335–342, 1974.

[CDS$^+$24] Alejandro Gomez Cadavid, Archismita Dalal, Anton Simen, Enrique Solano, and Narendra N. Hegade. Bias-field digitized counterdiabatic quantum optimization, 2024.

[CHH22] William Cook, Stephan Held, and Keld Helsgaun. Constrained local search for last-mile routing, 2022.

[CHP$^+$22] P. Chandarana, N. N. Hegade, K. Paul, F. Albarrán-Arriagada, E. Solano, A. del Campo, and Xi Chen. Digitized-counterdiabatic quantum approximate optimization algorithm. *Phys. Rev. Res.*, 4:013141, Feb 2022.

[Chu86] Fan RK Chung. Diameters of communication networks. *AMS Short Course on the Mathematics of Information Processing*, 34:1–18, 1986.

[Chu87] Fan RK Chung. Diameters of graphs: Old problems and new results. *Congressus Numerantium*, 60(2):295–317, 1987.

[Con79] George M Constantinides. Multiperiod consumption and investment behavior with convex transactions costs. *Manage. Sci.*, 25(11):1127–1137, November 1979.

[CP23] Cyrille W Combettes and Sebastian Pokutta. Revisiting the approximate carathéodory problem via the frank-wolfe algorithm. *Mathematical Programming*, 197(1):191–214, 2023.

[cpl25] CPLEX. https://www.ibm.com/products/ilog-cplex-optimization-studio, 2025. [Online; accessed April-3-2025].

[CPSP19] Pieter W. Claeys, Mohit Pandey, Dries Sels, and Anatoli Polkovnikov. Floquet-engineering counterdiabatic protocols in quantum many-body systems. *Phys. Rev. Lett.*, 123:090602, Aug 2019.

[CPT18] Gérard Cornuéjols, Javier Peña, and Reha Tütüncü. *Optimization Methods in Finance*. Cambridge University Press, 2 edition, 2018.

[CR83] Gary Chamberlain and Michael Rothschild. Arbitrage, Factor Structure, and Mean-Variance Analysis on Large Asset Markets. *Econometrica*, 51(5):1281–1304, 1983.

[cuO] Nvidia cuopt / rotuing. [Online; accessed July-18-2025].

[CW64] G. Clarke and J. W. Wright. Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research*, 12(4):568–581, 1964.

[D'a02] Gene D'avolio. The market for borrowing stock. *Journal of financial economics*, 66(2-3):271–306, 2002.

[dan] dano3mip. https://miplib.zib.de/instance_details_dano3mip.html. [Online; accessed Feb-10-2025].

[DdJ20] Gautam K. Das, Guilherme D. da Fonseca, and Ramesh K. Jallu. Efficient independent set approximation in unit disk graphs. *Discrete Applied Mathematics*, 280:63–70, 2020. Algorithms and Discrete Applied Mathematics (CALDAM 2016).

[DEH$^+$23] Maxime Dupont, Bram Evert, Mark J. Hodson, Bhuvanesh Sundar, Stephen Jeffrey, Yuki Yamaguchi, Dennis Feng, Filip B. Maciejewski, Stuart Hadfield, M. Sohaib Alam, Zhihui Wang, Shon Grabbe, P. Aaron Lott, Eleanor G. Rieffel, Davide Venturelli, and Matthew J. Reagor. Quantum-enhanced greedy combinatorial optimization solver. *Science Advances*, 9(45):eadi0487, 2023.

[DGS18] Iain Dunning, Swati Gupta, and John Silberholz. What works best when? A systematic evaluation of heuristics for Max-Cut and QUBO. *INFORMS J. Comput.*, 30(3):608–624, 2018.

[DGU07] Victor DeMiguel, Lorenzo Garlappi, and Raman Uppal. Optimal versus naive diversification: How inefficient is the 1/n portfolio strategy? *The Review of Financial Studies*, 22(5):1915–1953, 12 2007.

[DH99] Christoph Durr and Peter Hoyer. A quantum algorithm for finding the minimum, 1999.

[dim] DIMACS Implementation Challenges. http://dimacs.rutgers.edu/programs/challenge. [Online; accessed April-3-2025].

[DMBS79] Jack J Dongarra, Cleve Barry Moler, James R Bunch, and Gilbert W Stewart. *LINPACK users' guide*. SIAM, 1979.

[DR59] George B Dantzig and John H Ramser. The Truck Dispatching Problem. *Management science*, 6(1):80–91, 1959.

[DSE$^+$24] Maxime Dupont, Bhuvanesh Sundar, Bram Evert, David E. Bernal Neira, Zedong Peng, Stephen Jeffrey, and Mark J. Hodson. Quantum Optimization for the Maximum Cut Problem on a Superconducting Quantum Computer, 2024.

[DU16] Fanny Dufossé and Bora Uçar. Notes on birkhoff–von neumann decomposition of doubly stochastic matrices. *Linear Algebra and its Applications*, 497:108–115, 2016.

[dW81] D. de Werra. Scheduling in sports. In P. Hansen, editor, *Studies on graphs and discrete programming*, pages 381–395, Amsterdam, 1981. North-Holland.

[dW23] Ronald de Wolf. Quantum computing: Lecture notes, 2023.

[EJBB24] Manal El Jaouhari, Ghita Bencheikh, and Ghizlane Bencheikh. Exploring the capacitated vehicle routing problem using the power of machine learning: A literature review. In Youssef Benadada, Fatima-Zahra Mhada, Jaouad Boukachour, Fatima Ouzayd, and Ahmed El Hilali Alaoui, editors, *Proceeding of the 7th International Conference on Logistics Operations Management, GOL'24*, pages 68–80, Cham, 2024. Springer Nature Switzerland.

[EKC$^+$22] Sepehr Ebadi, Alexander Keesling, Madelyn Cain, Tout T Wang, Harry Levine, Dolev Bluvstein, Giulia Semeghini, Ahmed Omran, J-G Liu, Rhine Samajdar, et al. Quantum optimization of maximum independent set using Rydberg atom arrays. *Science*, 376(6598):1209–1215, 2022.

[ELL21] Sourour Elloumi, Amélie Lambert, and Arnaud Lazare. Solving unconstrained 0-1 polynomial programs through quadratic convex reformulation. *Journal of Global Optimization*, 80(2):231–248, 2021.

[EMW21] Daniel J. Egger, Jakub Mareček, and Stefan Woerner. Warm-starting quantum optimization. *Quantum*, 5:479, June 2021.

[EQSU23] Najib Errami, Eduardo Queiroga, Ruslan Sadykov, and Eduardo Uchoa. VRP-SolverEasy: A Python library for the exact solution of a rich vehicle routing problem. *INFORMS Journal on Computing*, 36:956–965, 2023.

[FGG14] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. A quantum approximate optimization algorithm, 11 2014.

[FGL$^+$21] David Fitzek, Toheed Ghandriz, Leo Laine, Mats Granath, and Anton Frisk Kockum. Applying quantum approximate optimization to the heterogeneous vehicle routing problem. *Scientific Reports*, 14, 2021.

[FHI$^+$21] Nils Froleyks, Marijn Heule, Markus Iser, Matti Järvisalo, and Martin Suda. Sat competition 2020. *Artificial Intelligence*, 301:103572, 2021.

[FHK$^+$21] Christoph Fraunholz, Dirk Hladik, Dogan Keles, Dominik Möst, and Wolf Fichtner. On the long-term efficiency of market splitting in Germany. *Energy Policy*, 149:111833, 2021.

[FM23] K. Fujii and T. Matsui. Solving break minimization problems in mirrored double round-robin tournament with QUBO solver, 2023.

[Fra] Frankwolfe. https://github.com/ZIB-IOL/FrankWolfe.jl. [Online; accessed April-3-2025].

[Fre56] J. E. Freund. Round Robin Mathematics. *The American Mathematical Monthly*, 63:112–114, 1956.

[FRG$^+$19] Sebastian Feld, Christoph Roch, Thomas Gabor, Christian Seidel, Florian Neukart, Isabella Galter, Wolfgang Mauerer, and Claudia Linnhoff-Popien. A Hybrid Solution Method for the Capacitated Vehicle Routing Problem Using a Quantum Annealer. *Frontiers in ICT*, 6:13, 2019.

[FRH$^+$22] Jernej Rudi Finžgar, Philipp Ross, Leonhard Hölscher, Johannes Klepsch, and Andre Luckow. Quark: A framework for quantum computing application benchmarking. In *2022 IEEE International Conference on Quantum Computing and Engineering (QCE)*, pages 226–237, 2022.

[Gav91] Bezalel Gavish. Topological design of telecommunication networks-local access design methods. *Annals of Operations Research*, 33(1):17–71, 1991.

[GC99] Milind Dawande Gérard Cornuéjols. Class of hard small 0-1 programs. *INFORMS Journal on Computing*, 11:205–210, 1999.

[GCF99] Bernard Gendron, Teodor Gabriel Crainic, and Antonio Frangioni. Multicommodity capacitated network design. In *Telecommunications network planning*, pages 1–19. Springer, 1999.

[GD25] Etienne Granet and Henrik Dreyer. AppQSim: Application-oriented benchmarks for Hamiltonian simulation on a quantum computer, 2025.

[GHG+21] Ambros Gleixner, Gregor Hendel, Gerald Gamrath, Tobias Achterberg, Michael Bastubbe, Timo Berthold, Philipp M. Christophel, Kati Jarck, Thorsten Koch, Jeff Linderoth, Marco Lübbecke, Hans Mittelmann, Derya Ozyurt, Ted Ralphs, Domenico Salvagnin, and Yuji Shinano. MIPLIB 2017: Data-Driven Compilation of the 6th Mixed-Integer Programming Library. *Mathematical Programming Computation*, 13(3):443 – 490, 2021.

[GJ79] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* Freeman, 1979.

[GKD18] Fred Glover, Gary Kochenberger, and Yu Du. A Tutorial on Formulating and Using QUBO Models. *arXiv preprint arXiv:1811.11538*, 2018.

[GKM+21] Aldy Gunawan, Graham Kendall, Barry McCollum, Hsin-Vonn Seow, and Lai Soon Lee. Vehicle routing: Review of benchmark datasets. *Journal of the Operational Research Society*, 72(8):1794–1807, 2021.

[GKVdW05] Alexei A Gaivoronski, Sergiy Krylov, and Nico Van der Wijst. Optimal portfolio selection and dynamic benchmark tracking. *European Journal of operational research*, 163(1):115–131, 2005.

[GM24] Jay Gambetta and Ryan Mandelbaum. IBM Quantum delivers on performance challenge made two years ago. https://www.ibm.com/quantum/blog/qdc-2024, 2024. [Online; accessed January-6-2025].

[GMS+16] Veronika Grimm, Alexander Martin, Martin Schmidt, Martin Weibelzahl, and Gregor Zöttl. Transmission and generation investment in electricity markets: The effects of market splitting and network fee regimes. *European Journal of Operational Research*, 254(2):493–509, 2016.

[GMW96a] Martin Grötschel, Alexander Martin, and Robert Weismantel. Packing Steiner trees: polyhedral investigations. *Mathematical Programming*, 72:101–123, 1996.

[GMW96b] Martin Grötschel, Alexander Martin, and Robert Weismantel. Packing Steiner trees: separation algorithms. *SIAM Journal on Discrete Mathematics*, 9(2):233–257, 1996.

[GMW97] Martin Grötschel, Alexander Martin, and Robert Weismantel. The Steiner tree packing problem in VLSI design. *Mathematical Programming*, 78:265–281, 1997.

[goo] Google or tools / routing. [Online; accessed July-18-2025].

[GP13] Nicolae Gârleanu and Lasse Heje Pedersen. Dynamic trading with predictable returns and transaction costs. *J. Finance*, 68(6):2309–2340, December 2013.

[GT22] Jérôme Gava and Julien Turc. The properties of alpha risk parity portfolios. *Entropy*, 24(11), 2022.

[gur25] Gurobi. https://www.gurobi.com/solutions/gurobi-optimizer/, 2025. [Online; accessed April-3-2025].

[GW95] Michel X. Goemans and David P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *J. ACM*, 42(6):1115–1145, nov 1995.

[GWG21] Austin Gilliam, Stefan Woerner, and Constantin Gonciulea. Grover adaptive search for constrained polynomial binary optimization. *Quantum*, 5:428, apr 2021.

[Haf80] L. R. Haff. Empirical Bayes Estimation of the Multivariate Normal Covariance Matrix. *The Annals of Statistics*, 8(3):586–597, 1980.

[Hak71] S Louis Hakimi. Steiner's problem in graphs and its implications. *Networks*, 1(2):113–133, 1971.

[HCS22] Narendra N Hegade, Xi Chen, and Enrique Solano. Digitized counterdiabatic quantum optimization. *Physical Review Research*, 4(4):L042030, 2022.

[Hel17] Keld Helsgaun. An Extension of the Lin-Kernighan-Helsgaun TSP Solver for Constrained Traveling Salesman and Vehicle Routing Problems. Technical report, Roskilde University, 2017.

[HGS] HGS-CVRP. https://github.com/vidalt/HGS-CVRP. [Online; accessed March-11-2025].

[HK12] Nam-Dũng Hoàng and Thorsten Koch. Steiner tree packing revisited. *Mathematical methods of operations research*, 76:95–123, 2012.

[HKRV11] S. Held, B. Korte, D. Rautenbach, and J. Vygen. Combinatorial optimization in VLSI design. In V. Chvátal, editor, *Combinatorial Optimization: Methods and Applications*, pages 33—96. IOS Press, 2011.

[HLH$^+$20] Ayaan Hossain, Eriberto Lopez, Sean M Halper, Daniel P Cetnar, Alexander C Reis, Devin Strickland, Eric Klavins, and Howard M Salis. Automated design of thousands of nonrepetitive parts for engineering stable genetic systems. *Nature biotechnology*, 38(12):1466–1475, 2020.

[HMCL24] James Holliday, Braeden Morgan, Hugh Churchill, and Khoa Luu. Hybrid Quantum Tabu Search for Solving the Vehicle Routing Problem, 2024.

[HNN$^+$20] Ramkumar Harikrishnakumar, Saideep Nannapaneni, Nam H. Nguyen, James E. Steck, and Elizabeth C. Behrman. A quantum annealing approach for dynamic multi-depot capacitated vehicle routing problem, 2020.

[HP23] Camilo Hernández and Dylan Possamaï. Me, myself and I: A general theory of non-Markovian time-inconsistent stochastic control for sophisticated agents. *The Annals of Applied Probability*, 33(2):1396 – 1458, 2023.

[HPD$^+$21] Narendra N Hegade, Koushik Paul, Yongcheng Ding, Mikel Sanz, Francisco Albarrán-Arriagada, Enrique Solano, and Xi Chen. Shortcuts to adiabaticity in digitized adiabatic quantum computing. *Physical Review Applied*, 15(2):024038, 2021.

[HS60] Alan J Hoffman and Robert R Singleton. On Moore graphs with diameters 2 and 3. *IBM Journal of Research and Development*, 4(5):497–504, 1960.

[HT21] Takuya Hatomura and Kazutaka Takahashi. Controlling and exploring quantum systems by algebraic expression of adiabatic gauge potential. *Phys. Rev. A*, 103:012220, Jan 2021.

[HTBP22] Deborah Hendrych, Hannah Troppens, Mathieu Besançon, and Sebastian Pokutta. Convex mixed-integer optimization with frank-wolfe methods. *arXiv preprint arXiv:2208.11010*, 2022.

[HZ21] Stefan Hougardy and Xianghui Zhong. Hard to solve instances of the euclidean traveling salesman problem. *Mathematical Programming Computation*, 13:51–74, 2021.

[I75] Rosenberg I. Reduction of bivalent maximization to the quadratic case. *Cah. centre et. rech. operat.*, 17(1):71–74, 1975.

[IBM] IBM Quantum. IBM Quantum Platform. https://quantum.ibm.com. [Online; accessed 17-February-2025].

[isk] Iskay Quantum Optimizer - A Qiskit Function by Kipu Quantum. https://docs.quantum.ibm.com/guides/docs.quantum.ibm.com/guides/kipu-optimization. [Online; accessed Feb-20-2025].

[ITC] International timetabling competition on sports timetabling 2021. https://robinxval.ugent.be/ITC2021/index.php. [Online; accessed 26-March-2025].

[Jag13] Martin Jaggi. Revisiting Frank-Wolfe: Projection-free sparse convex optimization. In Sanjoy Dasgupta and David McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 427–435, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR.

[Jai91] Raj Jain. *The Art of Computer Systems Performance Analysis: Techniques For Experimental Design, Measurement, Simulation, and Modeling.* Wiley, 04 1991.

[JATK$^+$24] Ali Javadi-Abhari, Matthew Treinish, Kevin Krsulich, Christopher J. Wood, Jake Lishman, Julien Gacon, Simon Martiel, Paul D. Nation, Lev S. Bishop, Andrew W. Cross, Blake R. Johnson, and Jay M. Gambetta. Quantum computing with Qiskit, 2024.

[JMS03] Kamal Jain, Mohammad Mahdian, and Mohammad R Salavatipour. Packing Steiner trees. In *SODA*, volume 3, pages 266–274, 2003.

[JSW$^+$24] Stephen P. Jordan, Noah Shutty, Mary Wootters, Adam Zalcman, Alexander Schmidhuber, Robbie King, Sergei V. Isakov, and Ryan Babbush. Optimization by decoded quantum interferometry, 2024.

[Kar72] Richard M. Karp. Reducibility among combinatorial problems. In Raymond E. Miller, James W. Thatcher, and Jean D. Bohlinger, editors, *Complexity of Computer Computations: Proceedings of a symposium on the Complexity of Computer Computations*, pages 85–103. Springer US, Boston, MA, 1972.

[KBPV22] Thorsten Koch, Timo Berthold, Jaap Pedersen, and Charlie Vanaret. Progress in mathematical programming solvers from 2001 to 2020. *EURO Journal on Computational Optimization*, 10:100031, 2022.

[KEA$^+$23] Youngseok Kim, Andrew Eddins, Sajant Anand, Ken Xuan Wei, Ewout van den Berg, Sami Rosenblatt, Hasan Nayfeh, Yantao Wu, Michael Zaletel, Kristan Temme, and Abhinav Kandala. Evidence for the utility of quantum computing before fault tolerance. *Nature*, 618(7965):500–505, Jun 2023.

[KFH$^+$15] Michihiro Koibuchi, Ikki Fujiwara, Shoichi Hirasawa, Ryota Yasudo, Satoshi Fujita, Koji Nakano, Takeaki Uno, Takeru Inoue, and Ken-ichi Kawarabayashi. Graph golf: the order/degree problem competition, 2015. Accessed on September 16, 2024.

[Kho02] Subhash Khot. On the power of unique 2-prover 1-round games. In *34th Annual ACM Symposium on Theory of Computing STOC'02*, pages 767–775, 2002.

[KK25] Nils-Christian Kempke and Thorsten Koch. A gpu accelerated variant of schroeppel-shamir's algorithm for solving the market split problem. Technical Report 25-10, ZIB, Takustr. 7, 14195 Berlin, 2025.

[KKMO07] Subhash Khot, Guy Kindler, Elchanan Mossel, and Ryan O'Donnell. Optimal Inapproximability Results for MAX-CUT and Other 2-Variable CSPs? *SIAM Journal on Computing*, 37(1):319–357, 2007.

[KKN22] M. Kuramata, R. Katsuki, and K. Nakata. Solving large break minimization problems in a mirrored double round-robin tournament using quantum annealing. *PLOS ONE*, e0266846:1–18, 2022.

[KLS17] Janardhan Kulkarni, Euiwoong Lee, and Mohit Singh. Minimum Birkhoff-von Neumann Decomposition. In Friedrich Eisenbrand and Jochen Koenemann, editors, *Integer Programming and Combinatorial Optimization*, pages 343–354, Cham, 2017. Springer International Publishing.

[KMV00] T. Koch, A. Martin, and S. Voß. SteinLib: An updated library on steiner tree problems in graphs. Technical Report ZIB-Report 00-37, Konrad-Zuse-Zentrum für Informationstechnik Berlin, Takustr. 7, Berlin, 2000.

[KÖ09] P. Kaski and P. RJ Östergård. There are 1,132,835,421,602,062,347 nonisomorphic one-factorizations of K_{14}. *J. Comb. Des.*, 17:147–159, 2009.

[Koc04] Thorsten Koch. *Rapid Mathematical Programming.* PhD thesis, Technische Universität Berlin, 2004. ZIB-Report 04-58.

[KPS19] Iordanis Kerenidis, Anupam Prakash, and Dániel Szilágyi. Quantum algorithms for portfolio optimization. In *Proceedings of the 1st ACM Conference on Advances in Financial Technologies*, pages 147–155, 2019.

[KR19] Petter N Kolm and Gordon Ritter. Modern perspectives on reinforcement learning in finance. *SSRN Electron. J.*, 2019.

[KSMP17] Michael Kolodrubetz, Dries Sels, Pankaj Mehta, and Anatoli Polkovnikov. Geometry and non-adiabatic response in quantum and classical systems. *Physics Reports*, 697:1–87, 2017.

[KTF14] Petter N. Kolm, Reha Tütüncü, and Frank J. Fabozzi. 60 Years of portfolio optimization: Practical challenges and current trends. *European Journal of Operational Research*, 234(2):356–371, 2014. 60 years following Harry Markowitz's contribution to portfolio theory and operations research.

[KvHW19] Wouter Kool, Herke van Hoof, and Max Welling. Attention, learn to solve routing problems!, 2019.

[LAB] LABS BF-DCQO Benchmark. https://github.com/AlejoKQ/data_LABS_BF-DCQO_Benchmark/tree/main. [Online; accessed April-3-2025].

[LAZ+25] Deep Lall, Abhishek Agarwal, Weixi Zhang, Lachlan Lindoy, Tobias Lindström, Stephanie Webster, Simon Hall, Nicholas Chancellor, Petros Wallden, Raul Garcia-Patron, Elham Kashefi, Viv Kendon, Jonathan Pritchard, Alessandro Rossi, Animesh Datta, Theodoros Kapourniotis, Konstantinos Georgopoulos, and Ivan Rungger. A review and collection of metrics and benchmarks for quantum computers: definitions, methodologies and software, 2025.

[LCBP99] Laurent Laloux, Pierre Cizeau, Jean-Philippe Bouchaud, and Marc Potters. Noise Dressing of Financial Correlation Matrices. *Phys. Rev. Lett.*, 83:1467–1470, Aug 1999.

[LCM+24] Thomas Lubinski, Carleton Coffrin, Catherine McGeoch, Pratik Sathe, Joshua Apanavicius, David Bernal Neira, Quantum Economic Development Consortium, et al. Optimization applications as quantum performance benchmarks, 2024.

[LdP16] Marcos López de Prado. Building diversified portfolios that outperform out of sample. *J. Portf. Manag.*, 42(4):59–69, May 2016.

[LDTA23] Ioannis D Leonidas, Alexander Dukakis, Benjamin Tan, and Dimitris G Angelakis. Qubit efficient quantum algorithms for the vehicle routing problem on nisq processors. *arXiv e-prints*, pages arXiv–2306, 2023.

[Leh60] Derrick H Lehmer. Teaching combinatorial tricks to a computer. *Combinatorial Analysis*, pages 179–193, 1960.

[Lev71] Anatoly Y. Levin. Algorithm for shortest connection of a group of graph vertices. *Soviet Mathematics Doklady*, 12:1477–1481, 1971.

[Lju21] Ivana Ljubić. Solving Steiner trees: Recent advances, challenges, and perspectives. *Networks*, 77(2):177–204, 2021.

[LK81] J. K. Lenstra and A. H. G. Rinnooy Kan. Complexity of vehicle routing and scheduling problems. *Networks*, 11(2):221–227, 1981.

[LKH] Lkh3. [Online; accessed July-18-2025].

[LLL+24] Fu Luo, Xi Lin, Fei Liu, Qingfu Zhang, and Zhenkun Wang. Neural combinatorial optimization with heavy decoder: Toward large scale generalization, 2024.

[LME+25] Jeanette Miriam Lorenz, Thomas Monz, Jens Eisert, Daniel Reitzner, Félicien Schopfer, Frédéric Barbaresco, Krzysztof Kurowski, Ward van der Schoot, Thomas Strohm, Jean Senellart, Cécile M. Perrault, Martin Knufinke, Ziyad Amodjee, and Mattia Giardini. Systematic benchmarking of quantum computers: status and recommendations, 2025.

[LML$^+$15] He Liu, Matthew K Mukerjee, Conglong Li, Nicolas Feltman, George Papen, Stefan Savage, Srinivasan Seshan, Geoffrey M Voelker, David G Andersen, Michael Kaminsky, et al. Scheduling techniques for hybrid circuit/packet networks. In *Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies*, pages 1–13, 2015.

[LOQ] Ivan Lima, Daniel Oliveira, and Eduardo Queiroga. Cvrplib.

[LR24] Debbie Lim and Patrick Rebentrost. A quantum online portfolio optimization algorithm. *Quantum Information Processing*, 23(3):63, 2024.

[Luc14] Andrew Lucas. Ising formulations of many NP problems. *Front. Phys.*, 2:5, 2014.

[LUM22] Xiaoyue Li, A. Sinem Uysal, and John M. Mulvey. Multi-period portfolio optimization using model predictive control with mean-variance and risk parity frameworks. *European Journal of Operational Research*, 299(3):1158–1176, 2022.

[MAA21] Simon Martiel, Thomas Ayral, and Cyril Allouche. Benchmarking Quantum Coprocessors in an Application-Centric, Hardware-Agnostic, and Scalable Way. *IEEE Transactions on Quantum Engineering*, 2:1–11, 2021.

[Mar52] Harry Markowitz. Portfolio Selection. *The Journal of Finance*, 7(1):77–91, 1952.

[MAT$^+$24] Nicola Mariella, Albert Akhriev, Francesco Tacchino, Christa Zoufal, Juan Carlos Gonzalez-Espitia, Benedek Harsanyi, Eugene Koskin, Ivano Tavernelli, Stefan Woerner, Marianna Rapsomaniki, Sergiy Zhuk, and Jannis Born. Quantum theory and application of contextual optimal transport, 2024.

[MB98] Stephan Mertens and Christine Bessenrodt. On the ground states of the bernasconi model. *Journal of Physics A: Mathematical and General*, 31(16):3731, 1998.

[MBF23] Nishikanta Mohanty, Bikash K Behera, and Christopher Ferrie. Analysis of the vehicle routing problem solved via hybrid quantum algorithms in the presence of noisy channels. *IEEE Transactions on Quantum Engineering*, 2023.

[MBHI$^+$95] M. V. Marathe, H. Breu, H. B. Hunt III, S. S. Ravi, and D. J. Rosenkrantz. Simple heuristics for unit disk graphs. *Networks*, 25(2):59–68, 1995.

[MCG24a] Ryan Mandelbaum, Antonio D. Corcoles, and Jay Gambetta. IBM's Big Bet on the Quantum-Centric Supercomputer. https://spectrum.ieee.org/ibm-quantum-computer-2668978269, 2024. [Online; accessed 11-November-2024].

[McG24b] Catherine McGeoch. How not to fool the masses when giving performance results for quantum computers, 2024.

[MDM24] Dihin Muriyatmoko, Arif Djunaidy, and Ahmad Muklason. Heuristics and metaheuristics for solving capacitated vehicle routing problem: An algorithm comparison. *Procedia Computer Science*, 234:494–501, 2024. Seventh Information Systems International Conference (ISICO 2023).

[MDS17] Pedro Munari, Twan Dollevoet, and Remy Spliet. A generalized formulation for vehicle routing problems, 2017.

[Mer69] Robert C. Merton. Lifetime portfolio selection under uncertainty: The continuous-time case. *The Review of Economics and Statistics*, 51(3):247–257, 1969.

[MGJ$^+$12] Easwar Magesan, Jay M. Gambetta, B. R. Johnson, Colm A. Ryan, Jerry M. Chow, Seth T. Merkel, Marcus P. da Silva, George A. Keefe, Mary B. Rothwell, Thomas A. Ohki, Mark B. Ketchen, and M. Steffen. Efficient measurement of quantum gate error by interleaved randomized benchmarking. *Phys. Rev. Lett.*, 109:080505, Aug 2012.

[MHP$^+$23] David C. McKay, Ian Hincks, Emily J. Pritchett, Malcolm Carroll, Luke C. G. Govia, and Seth T. Merkel. Benchmarking quantum processor performance at scale, 2023.

[min24] MINLPLib: A library of mixed-integer and continuous nonlinear programming instances. https://www.minlplib.org/index.html, 2024. [Online; accessed April-3-2025].

[Mit23] Hans Mittelmann. Decision Tree for Optimization Software. https://plato.asu.edu/bench.html, 2023. [Online; accessed April-3-2025].

[MKS$^+$22] Samuel Mugel, Carlos Kuchkovsky, Escolastico Sanchez, Samuel Fernandez-Lorenzo, Jorge Luis-Hita, Enrique Lizaso, and Roman Orus. Dynamic portfolio optimization with real datasets using quantum processors and quantum-inspired tensor networks. *Physical Review Research*, 4(1):013006, 2022.

[MM08] Richard O. Michaud and Robert O. Michaud. *Efficient Asset Management: A Practical Guide to Stock Portfolio Optimization and Asset Allocation*. Number 9780195331912 in OUP Catalogue. Oxford University Press, 2008.

[MM15] Richard O. Michaud and Tongshu Ma. Efficient Asset Management: A Practical Guide to Stock Portfolio Optimization and Asset Allocation. *The Review of Financial Studies*, 14(3):901–904, 06 2015.

[MMW86] Thomas L Magnanti, Paul Mireault, and Richard T Wong. *Tailoring Benders decomposition for uncapacitated network design*. Springer, 1986.

[MR59] Marvin Marcus and Rimhak Ree. Diagonals of doubly stochastic matrices. *The Quarterly Journal of Mathematics*, 10(1):296–302, 1959.

[MS12] Mirka Miller and Jozef Sirán. Moore graphs and beyond: A survey of the degree/diameter problem. *The electronic journal of combinatorics*, pages DS14–May, 2012.

[MTZ60] C. E. Miller, A. W. Tucker, and R. A. Zemlin. Integer programming formulation of traveling salesman problems. *J. ACM*, 7(4):326–329, 10 1960.

[MW84] Thomas L Magnanti and Richard T Wong. Network design and transportation planning: Models and algorithms. *Transportation science*, 18(1):1–55, 1984.

[MWHK11] Matthias Müller, Brian Whitney, Robert Henschel, and Kalyan Kumaran. *SPEC Benchmarks*, pages 1886–1893. Springer US, Boston, MA, 2011.

[NSB$^+$24] Paul D. Nation, Abdullah Ash Saki, Sebastian Brandhofer, Luciano Bello, Shelly Garion, Matthew Treinish, and Ali Javadi-Abhari. Benchmarking the performance of quantum computing software, 2024.

[NT98] G. L. Nemhauser and M. A. Trick. Scheduling a major college basketball conference. *Oper. Res.*, 46:1–8, 1998.

[NTI$^+$23] Koji Nakano, Daisuke Takafuji, Yasuaki Ito, Takashi Yazane, Junko Yano, Shiro Ozaki, Ryota Katsuki, and Rie Mori. Diverse adaptive bulk search: a framework for solving qubo problems on multiple gpus. In *2023 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 314–325, 2023.

[OFCCBM23] Corey O'Meara, Marina Fernández-Campoamor, Giorgio Cortiana, and Juan Bernabé-Moreno. Quantum software architecture blueprints for the cloud: Overview and application to peer-2-peer energy trading. In *2023 IEEE Conference on Technologies for Sustainability (SusTech)*, pages 191–198. IEEE, 2023.

[PBE24] Elijah Pelofske, Andreas Bärtschi, and Stephan Eidenbenz. Short-depth QAOA circuits and quantum annealing on higher-order Ising models. *npj Quantum Inf.*, 10(1):30, 2024.

[PM16] Tom Packebusch and Stephan Mertens. Low autocorrelation binary sequences. *Journal of Physics A: Mathematical and Theoretical*, 49(16):165001, 2016.

[Pol03] Tobias Polzin. *Algorithms for the Steiner problem in networks*. PhD thesis, Universität des Saarlandes, 2003.

[PPSHW15] Jan Palczewski, Rolf Poulsen, Klaus Reiner Schenk-Hoppé, and Huamao Wang. Dynamic portfolio optimization with transaction costs and state-dependent drift. *European Journal of Operational Research*, 243(3):921–931, 2015.

[Pra19] Marcos Prado. A Robust Estimator of the Efficient Frontier. *SSRN Electronic Journal*, 01 2019.

[PSUV20] Artur Pessoa, Ruslan Sadykov, Eduardo Uchoa, and François Vanderbeck. A generic exact solver for vehicle routing and related problems. *Math. Programming*, 183:483–523, 2020.

[PYBBK24] Timothy Proctor, Kevin Young, Andrew D. Baczewski, and Robin Blume-Kohout. Benchmarking quantum computers, 2024.

[QAO] Independent set benchmarking. https://github.com/eggerdj/independent_set_benchmarking. [Online; accessed 26-March-2025].

[Qis] Qiskit Ecosystem: Qiskit Optimization. https://github.com/qiskit-community/qiskit-optimization. [Online; accessed 31-March-2025].

[QOBLIB] Quantum Optimization Benchmarking Library. https://git.zib.de/qopt/qoblib-quantum-optimization-benchmarking-library/. [Online; accessed 1-April-2025].

[QUB25] QUBO++ with ABS2 GPU QUBO Solver. https://abs2.cs.hiroshima-u.ac.jp/home, 2025. [Online, accessed 31-March-2025].

[RA00] Neil Chriss Robert Almgren. Optimal execution of portfolio transactions. *Journal of Risk*, 3(2):5–39, 2000.

[RA15] Ryan A. Rossi and Nesreen K. Ahmed. The network data repository with interactive graph analytics and visualization. In *AAAI*, 2015.

[RHG$^+$16] Gili Rosenberg, Poya Haghnegahdar, Phil Goddard, Peter Carr, Kesheng Wu, and Marcos López de Prado. Solving the optimal trading trajectory problem using a quantum annealer. *IEEE Journal of Selected Topics in Signal Processing*, 10(6):1053–1060, 2016.

[Riv24] Fabian Rivetta. Solving maximum independent set problems and analysis of the influence of various constraint classes. B.sc. thesis, Technische Universität Berlin, 2024. B.Sc. Thesis - Technische Universität Berlin supervised by Thorsten Koch.

[RK23] Daniel Rehfeldt and Thorsten Koch. Implications, Conflicts, and Reductions for Steiner Trees. *Mathematical Programming*, 197:903 – 966, 2023.

[RPDGS22] R. M. Rosati, M. Petris, L. Di Gaspero, and A. Schaerf. Multi-neighborhood simulated annealing for the sports timetabling competition ITC2021. *J. Sched.*, 25:301–319, 2022.

[RT95] Yves Rochat and Éric D. Taillard. Probabilistic diversification and intensification in local search for vehicle routing. *Journal of Heuristics*, 1(1):147–167, Sep 1995.

[RVC$^+$24] Sebastián V Romero, Anne-Maria Visuri, Alejandro Gomez Cadavid, Enrique Solano, and Narendra N Hegade. Bias-field digitized counterdiabatic quantum algorithm for higher-order binary optimization. *arXiv preprint arXiv:2409.04477*, 2024.

[SAT] The International SAT Competition Web Page. https://satcompetition.github.io.

[Sch70] Manfred Schroeder. Synthesis of low-peak-factor signals and binary sequences with low autocorrelation (corresp.). *IEEE transactions on Information Theory*, 16(1):85–89, 1970.

[Sch03] Alexander Schrijver. *Combinatorial Optimization*. Springer, 2003.

[SFE$^+$23] Reza Saberi, Hamid Falaghi, Mostafa Esmaeeli, Maryam Ramezani, Ali Ashoornezhad, and Reza Izadpanah. Power distribution network expansion planning to improve resilience. *IET Generation, Transmission & Distribution*, 17(21):4701–4716, 2023.

[SFT16] Abu Bakar Siddique, Saadia Farid, and Muhammad Tahir. Proof of bijection for combinatorial number system. *arXiv preprint arXiv:1601.05794*, 2016.

[SHPG12] Ankit Singla, Chi-Yao Hong, Lucian Popa, and P Brighten Godfrey. Jellyfish: Networking data centers randomly. In *9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*, pages 225–238, 2012.

[SJHE24] Gopal Chandra Santra, Fred Jendrzejewski, Philipp Hauke, and Daniel J. Egger. Squeezing and quantum approximate optimization. *Phys. Rev. A*, 109:012413, Jan 2024.

[SLC$^+$24] Ruslan Shaydulin, Changhao Li, Shouvanik Chakrabarti, Matthew DeCross, Dylan Herman, Niraj Kumar, Jeffrey Larson, Danylo Lykov, Pierre Minssen, Yue Sun, Yuri Alexeev, Joan M. Dreiling, John P. Gaebler, Thomas M. Gatterman, Justin A. Gerber, Kevin Gilmore, Dan Gresh, Nathan Hewitt, Chandler V. Horst, Shaohan Hu, Jacob Johansen, Mitchell Matheny, Tanner Mengle, Michael Mills, Steven A. Moses, Brian Neyenhuis, Peter Siegfried, Romina Yalovetzky, and Marco Pistoia. Evidence of scaling advantage for the quantum approximate optimization algorithm on a classically intractable problem. *Science Advances*, 10(22):eadm6761, 2024.

[Slo25] N. J. A. Sloane. Challenge problems: Independent sets in graphs. Online database entry, 2025. Accessed: 2025-02-25.

[SM21] Konstantin Sidorov and Alexander Morozov. A review of approaches to modeling applied vehicle routing problems, 2021.

[SMMn23] K. Smith-Miles and M. A. Muñoz. Instance space analysis for algorithm testing: Methodology and software tools. *ACM Comput. Surv.*, 55, 2023.

[SMMW21] Nicholas Slate, Edric Matwiejew, Samuel Marsh, and Jingbo B Wang. Quantum walk-based portfolio optimisation. *Quantum*, 5:513, 2021.

[SS81] Richard Schroeppel and Adi Shamir. A $t = O(2^{n/2})$, $s = O(2^{n/4})$ algorithm for certain NP-complete problems. *SIAM Journal on Computing*, 10(3):456–464, 1981.

[SWS11] Ji-Yong Shin, Bernard Wong, and Emin Gün Sirer. Small-world datacenters. In *Proceedings of the 2nd ACM Symposium on Cloud Computing*, pages 1–13, 2011.

[Tep00] Lucie Tepla. Optimal portfolio policies with borrowing and shortsale constraints. *Journal of Economic Dynamics and Control*, 24(11-12):1623–1639, 2000.

[TGO$^+$22] T. Tomesh, P. Gokhale, V. Omole, G. Ravi, K. N. Smith, J. Viszlai, X. Wu, N. Hardavellas, M. R. Martonosi, and F. T. Chong. SupermarQ: A Scalable Quantum Benchmark Suite. In *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 587–603, Los Alamitos, CA, USA, apr 2022. IEEE Computer Society.

[TK04] R.H. Tütüncü and M. Koenig. Robust asset allocation. *Annals of Operations Research*, 132(1-4):157 – 187, 2004.

[TV02a] Paolo Toth and Daniele Vigo. Models, relaxations and exact approaches for the capacitated vehicle routing problem. *Discrete Applied Mathematics*, 123(1):487–512, 2002.

[TV02b] Paolo Toth and Daniele Vigo. *The Vehicle Routing Problem.* Society for Industrial and Applied Mathematics, 2002.

[TZX$^+$21] Haisheng Tan, Chi Zhang, Chao Xu, Yupeng Li, Zhenhua Han, and Xiang-Yang Li. Regularization-based coflow scheduling in optical circuit switches. *IEEE/ACM Transactions on Networking*, 29(3):1280–1293, 2021.

[UPP$^+$17] Eduardo Uchoa, Diego Pecin, Artur Pessoa, Marcus Poggi, Thibaut Vidal, and Anand Subramanian. New benchmark instances for the capacitated vehicle routing problem. *European Journal of Operational Research*, 257(3):845–858, 2017.

[UYS18] H. L. Urdaneta, J. Yuan, and A. S. Siqueira. Alternative integer linear and quadratic programming formulations for ha-assignment problems. *Proceeding Series of the Brazilian Society of Computational and Applied Mathematics*, 6, 2018.

[VBG23a] D. Van Bulck and D. Goossens. A traditional Benders' approach to sports timetabling. *Eur. J. Oper. Res.*, 307:813–826, 2023.

[VBG23b] D. Van Bulck and D. Goossens. The international timetabling competition on sports timetabling (ITC2021). *Eur. J. Oper. Res.*, 308:1249–1267, 2023.

[VG23] David Van Bulck and Dries Goossens. The international timetabling competition on sports timetabling (itc2021). *European Journal of Operational Research*, 308(3):1249–1267, 2023.

[VGC$^+$24] D. Van Bulck, D. Goossens, J.P. Clarner, A. Dimitsas, G. H.G. Fonseca, C. Lamas-Fernandez, M. M. Lester, J. Pedersen, A. E. Phillips, and R. M. Rosati. Which algorithm to select in sports timetabling? *European Journal of Operational Research*, 318:575–591, 2024.

[VGMG23] Eleonora Vercesi, Stefano Gualandi, Monaldo Mastrolilli, and Luca Maria Gambardella. On the generation of metric tsp instances with a large integrality gap by branch-and-cut. *Mathematical Programming Computation*, 15:389–416, 2023.

[VGO$^+$20] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.

[vHLS23] J. van Doornmalen, C. Hojny, R. Lambers, and F. C. R. Spieksma. Integer programming models for round robin tournaments. *Eur. J. Oper. Res.*, 310:24–33, 2023.

[Vid22] Thibaut Vidal. Hybrid genetic search for the CVRP: Open-source implementation and SWAP* neighborhood. *Computers & Operations Research*, 140:105643, 2022.

[VIT21] Víctor Valls, George Iosifidis, and Leandros Tassiulas. Birkhoff's decomposition revisited: Sparse scheduling for high-speed circuit switches. *IEEE/ACM Transactions on Networking*, 29(6):2399–2412, 2021.

[VLM20] Thibaut Vidal, Gilbert Laporte, and Piotr Matl. A concise guide to existing and emerging vehicle routing problem variants. *European Journal of Operational Research*, 286(2):401–416, 2020.

[Vog12] Heiko Vogel. Solving market split problems with heuristical lattice reduction. *Annals of Operations Research*, 196:581–590, 2012.

[VQG$^+$24] Deborah Volpe, Nils Quetschlich, Mariagrazia Graziano, Giovanna Turvani, and Robert Wille. Towards an automatic framework for solving optimization problems with quantum computers. In *2024 IEEE International Conference on Quantum Software (QSW)*, page 46–57. IEEE, jul 2024.

[VRPa] https://vrpsolver.math.u-bordeaux.fr. [Online; accessed 11-March-2025].

[VRPb] https://github.com/inria-UFF/VRPSolverEasy. [Online; accessed 11-March-2025].

[Wan21] Robert Waniek. Graph Golf: Betrachtung des Order-Degree-Problems. Master's thesis, Technische Universität Berlin, 2021.

[Was02] Alfred Wassermann. Attacking the market split problem with lattice point enumeration. *Journal of Combinatorial Optimization*, 6:5–16, 2002.

[Was25] Alfred Wassermann, 2025. private communication, paper in preparation.

[WLG$^+$22] Jonathan Wurtz, Pedro LS Lopes, Christoph Gorgulla, Nathan Gemelke, Alexander Keesling, and Shengtao Wang. Industry applications of neutral-atom quantum computing solving independent set problems. *arXiv preprint arXiv:2205.08500*, 2022.

[WPJA$^+$21] A. Wack, H. Paik, A. Javadi-Abhari, P. Jurcevic, I. Faro, J. M. Gambetta, and B. R. Johnson. Quality, speed, and scale: three key attributes to measure the performance of near-term quantum computers, 2021.

[WRV$^+$24] Dian Wu, Riccardo Rossi, Filippo Vicentini, Nikita Astrakhantsev, Federico Becca, Xiaodong Cao, Juan Carrasquilla, Francesco Ferrari, Antoine Georges, Mohamed Hibat-Allah, Masatoshi Imada, Andreas M. Läuchli, Guglielmo Mazzola, Antonio Mezzacapo, Andrew Millis, Javier Robledo Moreno, Titus Neupert, Yusuke Nomura, Jannes Nys, Olivier Parcollet, Rico Pohle, Imelda Romero, Michael Schmid, J. Maxwell Silvester, Sandro Sorella, Luca F. Tocchio, Lei Wang, Steven R. White, Alexander Wietek, Qi Yang, Yiqi Yang, Shiwei Zhang, and Giuseppe Carleo. Variational benchmarks for quantum many-body problems. *Science*, 386(6719):296–301, October 2024.

[WSW24] Jonathan Wurtz, Stefan Sack, and Sheng-Tao Wang. Solving non-native combinatorial optimization problems using hybrid quantum-classical algorithms, 2024.

[WVG$^+$22] Johannes Weidenfeller, Lucia C. Valor, Julien Gacon, Caroline Tornow, Luciano Bello, Stefan Woerner, and Daniel J. Egger. Scaling of the quantum approximate optimization algorithm on superconducting qubit based hardware. *Quantum*, 6:870, dec 2022.

[XLC$^+$24] Ningyi Xie, Xinwei Lee, Dongsheng Cai, Yoshiyuki Saito, Nobuyoshi Asai, and Hoong Chuin Lau. A feasibility-preserved quantum approximate solver for the capacitated vehicle routing problem. *Quantum Information Processing*, 23(8):291, 2024.

[XN17] Mingyu Xiao and Hiroshi Nagamochi. Exact algorithms for maximum independent set. *Information and Computation*, 255:126–146, 2017.

[XRA$^+$23] Pedro Maciel Xavier, Pedro Ripper, Tiago Andrade, Joaquim Dias Garcia, Nelson Maculan, and David E. Bernal Neira. QUBO.jl: A julia ecosystem for quadratic unconstrained binary optimization, 2023.

[XRP$^+$24] Pedro Maciel Xavier, Pedro Ripper, Joshua Pulsipher, Joaquim Dias Garcia, Nelson Maculan, and David E Bernal Neira. Disjunctive programming meets QUBO. In *Computer Aided Chemical Engineering*, volume 53, pages 3433–3438. Elsevier, 2024.

[YZLH17] Tong Ye, Jinghui Zhang, Tony T. Lee, and Weisheng Hu. Deflection-compensated birkhoff–von-neumann switches. *IEEE/ACM Transactions on Networking*, 25(2):879–895, 2017.

[ZSKP25] Zhiwei Zhang, Jiayu Shen, Niraj Kumar, and Marco Pistoia. New improvements in solving large labs instances using massively parallelizable memetic tabu search, 2025.