Towards an Automatic Framework for Solving Optimization Problems with Quantum Computers

Deborah Volpe*, Nils Quetschlich†, Mariagrazia Graziano‡, Giovanna Turvani*, and Robert Wille†§

*Department of Electronics and Telecommunications, Politecnico di Torino Italy
†Chair for Design Automation, Technical University of Munich, Germany
‡Department of Applied Science and Technology, Politecnico di Torino Italy
§Software Competence Center Hagenberg GmbH (SCCH), Austria
deborah.volpe@polito.it, nils.quetschlich@tum.de, mariagrazia.graziano@polito.it,
giovanna.turvani@polito.it and robert.wille@tum.de

Abstract—**Optimizing objective functions stands to benefit significantly from leveraging quantum computers, promising enhanced solution quality across various application domains in the future. However, harnessing the potential of quantum solvers necessitates formulating problems according to the *Quadratic Unconstrained Binary Optimization* (QUBO) model, demanding significant expertise in quantum computation and QUBO formulations. This expertise barrier limits access to quantum solutions.**

Fortunately, automating the conversion of conventional optimization problems into QUBO formulations presents a solution for promoting accessibility to quantum solvers. This article addresses the unmet need for a comprehensive automatic framework to assist users in utilizing quantum solvers for optimization tasks while preserving interfaces that closely resemble conventional optimization practices. The framework prompts users to specify variables, optimization criteria, as well as validity constraints and, afterwards, allows them to choose the desired solver. Subsequently, it automatically transforms the problem description into a format compatible with the chosen solver and provides the resulting solution. Additionally, the framework offers instruments for analyzing solution validity and quality.

Comparative analysis against existing libraries and tools in the literature highlights the comprehensive nature of the proposed framework. Two use cases (the knapsack problem and linear regression) are considered to show the completeness and efficiency of the framework in real-world applications.

Finally, the proposed framework represents a significant advancement towards automating quantum computing solutions and widening access to quantum optimization for a broader range of users.

The framework is publicly available on GitHub (https://github.com/cda-tum/mqt-qao) as part of the Munich Quantum Toolkit (MQT).

Index Terms—**QUBO, Quantum Computing, Design Automation, Quantum Optimization, Quantum Annealer, Quantum Approximate Optimization Algorithm, Variational Quantum Eigensolver, Grover Adaptive Search**

I. INTRODUCTION

Quantum computers have the potential to enhance the optimization of *objective functions* in specific application domains such as machine learning [1], [2], scheduling [3], and resource allocation [4]. However, leveraging *quantum solvers* requires writing the problem in a suitable format, demanding proficiency in quantum computers and quantum-compliant problem formulations. This challenges researchers or industries not directly engaged in quantum computation, inhibiting the initial exploration of quantum solutions for real-world use cases,

such as linear regression [1]. Automating the transformation of problems into a quantum-compliant format offers a solution to this obstacle.

This article proposes a framework to assist *non-experts* in exploring the *potential of quantum solvers* for *optimization problems*. It keeps the interfaces for the problem specification as similar as possible to conventional optimization. Users are prompted to declare variables and their operational ranges, optimization criteria, as well as validity constraints, and afterwards, allows them to choose the desired solver. Subsequently, the tool automatically transforms the problem description into a compatible format for the selected solver and delivers the obtained solution. Moreover, it offers mechanisms for evaluating solution validity and quality.

The framework's characteristics are compared to those of libraries and analogous tools already in the literature, distinguishing itself for its *comprehensiveness*. To this end, two use cases (knapsack problems and linear regression models) are considered to demonstrate the framework's effectiveness and flexibility in real-world application contexts. Notable benefits are observed from the user's perspective in both cases. Consequently, this framework, which is publicly available on GitHub (https://github.com/cda-tum/mqt-qao) as part of the Munich Quantum Toolkit (MQT) [5], represents an important step toward automating quantum computing solutions.

The rest of the article is organized as follows. Section II briefly reviews quantum optimization, focusing on quantum-compliant problem formulations, and discusses the current workflow for solving an optimization problem with a quantum solver. Based on that, Section III outlines the automation opportunities, which constitute the motivation behind the framework, and a comparison with the state-of-the-art. The actual implementation and its characteristics are described in Section IV. The use cases are presented in Section V, evaluating the framework from a user perspective and discussing its effectiveness and results. Finally, in Section VI, conclusions are drawn, and future perspectives are illustrated.

II. SOLVING OPTIMIZATION PROBLEMS WITH QUANTUM COMPUTERS

This section provides a comprehensive overview of quantum optimization, focusing on solvers and the QUBO formulation,

which is recognized as the best model for exploiting quantum computing. Additionally, the key steps required for leveraging quantum computers to solve optimization problems are outlined. In particular, stages demanding quantum and QUBO expertise are emphasized since the proposed framework aims to automate these.

A. Overview

In order to exploit the quantum computing paradigm for optimization problems, there are two possibilities: employing a *Quantum Annealer* (QA)—a *special-purpose quantum computer* theorized in 1998 [6]–[10], using the natural properties of a quantum system evolution for obtaining the ground state—or executing algorithms entirely or partially on a *general-purpose quantum computer compliant with the quantum circuit model* [11].

In the second case, nowadays, the quantum computer is usually exploited for accelerating specific tasks also involving classical computers. The most popular approaches in this context are:

- *Quantum Approximate Optimization Algorithm* (QAOA, [12], [13]), which is a hybrid quantum-classical algorithm that approximates the adiabatic evolution of a quantum system encoding the optimization problem in the Hilbert space;
- *Variational Quantum Eigensolver* (VQE, [14], [15]), which is a hybrid quantum-classical algorithm for identifying the lower energy eigenstate of a given physical system, allowing its applicability in a combinatorial optimization context;
- *Grover Adaptive Search* (GAS, [16]–[20]), a hybrid quantum-classical algorithm, which implements a successive approximation method for estimating the optimal value. It executes the Grover Search algorithm for sampling negative values of the problem cost function, iteratively moving up the classically obtained sample until the last negative value is found.

The formulation commonly employed for solving optimization problems with quantum computers is the *QUBO* model [21], [22]. The acronym stands for *Quadratic Unconstrained Binary Optimization*, where *quadratic* indicates the highest degree of the objective function, *unconstrainted* means that the constraints cannot be considered conventionally, *binary* signifies that only unipolar binary variables can be considered, and *optimization* emphasizes the purpose of the model. The associated problem objective function can be written as

$$f(\mathbf{b}) = \gamma + \sum_i \alpha_i \cdot b_i + \sum_{i<j} \beta_{ij} \cdot b_i b_j, \qquad (1)$$

where $b_i \in [0,1]$ is a binary variable, $b_i b_j$ is a coupler that allows two variables to influence each other, α_i is a weight or bias associated with a single variable, β_{ij} is a strength that controls the influence of variables i and j, and γ is an offset that can be neglected during optimization. Usually, this model is employed considering the *minimization* as the optimization direction.

This formulation can consider a constraint through the *aggregation method*. In particular, a *quadratic penalty function* $g(x)$—assuming value 0 for configurations satisfying the constraints and a positive amount otherwise—is included in the objective function, as shown by

$$\text{minimize} \quad y = f(\mathbf{b}) + \lambda g(\mathbf{b}), \qquad (2)$$

where λ is a positive penalty weight assigned to the constraint quadratic function $g(\mathbf{b})$.

Note that the QUBO formulation is also compatible with some classical solvers like *Simulated Annealing* (SA [23]). This algorithm is commonly employed for the initial validation of a QUBO formulation and as an alternative when the problem size exceeds the capabilities of current *Noisy Intermediate-Scale Quantum* (NISQ) devices and simulators since it can handle thousands of binary variables on classical machines.

Remembering these concepts, the rest of the section covers the steps required for solving an optimization problem with quantum computers.

B. Problem Specification

The first step for leveraging the optimization procedure is the declaration of the *problem specifications*, which consist of the degrees of freedom (i.e., the variables), the criteria for optimization (described as objective functions), and any requirements (also called constraints) that a valid solution must meet. The problem *variables* can be *binary* (unipolar or bipolar), *discrete* (specifying the valid values), and *continuous* (defining the operative range). Meanwhile, the objective function is a parametric description of a figure of merit whose optimal value could be its minimum or maximum, requiring the specification of the *optimization direction*. In the case of a problem composed of more than one expression to optimize (multi-objective optimization), a *criterion* is essential to establish the relative importance of each contribution in relation to the others.

These operations do not depend on the problem and the solver, thus maintaining consistency whether moving from classical to quantum optimization context. In the following, two examples, considered representative case studies throughout the remainder of this article, are introduced to illustrate this. The former is characterized by inequality constraints to handle, while the latter involves real variables. Together, they provide a wide overview of possible real-case scenarios.

Example 1. *The knapsack problem is a practical challenge where the objective is to determine the optimal subset S from a collection of N_{obj} objects to put into a bag. Each object is characterized by its weight w_{arr_i} and preference score p_{arr_i}. The optimal subset maximizes the total score, i.e., the sum of the scores of the taken object $\sum_{obj_i \in S} p_{arr_i}$, and ensures that the total weight $\sum_{obj_i \in S} w_{arr_i}$ does not exceed the maximum W_{max}.*
The problem variables represent the objects themselves, and the optimization goal is to maximize the total score. Additionally, a valid solution must adhere to the constraint that the total weight does not exceed W_{max}.

Example 2. *Linear regression seeks to determine the straight line that best fits a given dataset by minimizing the Euclidean error function:*

$$E(w) = \|Xw + Y\| = w^T X^T X^T X w - 2w^T X^T Y + Y^T Y \,,$$

where $X \in \mathbb{R}^{N \times (d+1)}$ is the augmented regression data matrix, N the number of data points in the training set, d the number of features, $Y \in \mathbb{R}^N$ is the regression label of the training data, $w \in \mathbb{R}^{d+1}$ the regression weights and $E(w)$ the Euclidian error function.
The problem variables are the regression weights $w \in \mathbb{R}^{d+1}$, and the optimization objective is to minimize the Euclidean error function. This problem is unconstrained.

C. Solver Selection

After the problem specification stage, the solving approach has to be chosen among QA, QAOA, VQE, and GAS. Subsequently, accurate consideration must be given to selecting appropriate settings for the chosen solver. For instance, choosing an appropriate annealing time for the problem of interest is crucial when QA is selected.

D. Encoding the Problem

After the solver selection, the problem must be written in a compliant format. As previously discussed, all the mentioned approaches support the QUBO formulation, which involves exclusively *binary variables*. Therefore, continuous and discrete variables must be expressed as a weighted set of binary ones through proper *encoding mechanisms*. Additionally, it requires including the constraints in the objective function through *weighted penalties* for allowing their evaluation during the optimization procedure, as explained in Section II-A.

Moreover, the quantum solver can only handle the minimization problem, requiring a sign change in the case of problems with a maximization objective, and solve *second-order polynomials*, necessitating a polynomial reduction step.

In the case of *multi-objective* optimization, the *aggregation* approach can be employed by combining objective functions into a higher scalar one, expressing a preference criterion.

Again, these steps are illustrated by considering the two running examples.

Example 1. *When formulating a knapsack problem in QUBO form, assigning a binary variable to each object is necessary. This step is straightforward in this case, as the problem naturally lends itself to a binary representation of variables. Additionally, optimization involves expressing the objective as a unified cost function, incorporating constraints as penalty functions. Translating inequality constraints into penalty functions requires considerable expertise in QUBO formulation, especially due to the need for auxiliary variables.*

Example 2. *The initial step for writing a linear regression problem into a QUBO form involves encoding the variables into a binary format. This process requires a suitable encoding mechanism since the original variables are real-valued. Subsequently, the expression must be redefined as a function of these binary variables.*

E. Solving the Problem

The resulting formulation is then submitted to the chosen solver with a proper *parameters configuration*. Due to their stochastic nature, these solvers are usually run multiple times, and the best-obtained result is considered.

Access to real quantum devices is enabled through cloud services using a dedicated account. Alternatively, QAOA, VQE, and GAS can also be executed on classical simulators, such as those referenced in [24]–[29].

F. Solution Analysis

Finally, the obtained solution must be *decoded*, the original problem variables recovered, and its quality evaluated. This process consists of evaluating the initial cost functions with the found configuration and verifying the satisfaction of constraints.

III. TOWARDS AN AUTOMATIC FRAMEWORK FOR SOLVING OPTIMIZATION PROBLEMS

This section presents the unmet needs that the proposed framework seeks to address, discusses the idea behind it, and offers a high-level description of its structure. Additionally, it compares the framework with existing state-of-the-art libraries and tools.

A. Motivations

All the steps reviewed above demand significant expertise in QUBO formulation and quantum solvers, which is uncommon among conventional optimization users. This expertise gap limits accessibility to the potential quantum computational advantage. Moreover, some tasks, such as variable encoding, solution decoding, as well as function rewriting can be cumbersome, and *manual execution* may introduce *errors*. Managing penalty functions is also critical, especially for translating constraints, which is not always effortless, especially when *auxiliary variables* are needed, e.g., for inequality constraints. Furthermore, determining an *appropriate penalty weight* is crucial for obtaining good results. A too-low value may not adequately penalize invalid solutions, while a too-high value may flatten the optimization function, complicating effective solution space exploration. Lastly, the configuration and the execution of solvers might not be intuitive for non-experts in quantum computing. For all these reasons, these steps can greatly benefit from *automation* for both *preventing errors* arising from the manual translation of the optimization problem into a solver-compatible format and *improving accessibility* to quantum solvers for a wider range of users.

B. General Idea

This work aims to offer a versatile and *user-friendly* framework for assisting non-experts in quantum solvers exploitation for real-world problems, keeping the user in the conventional optimization domain by requiring as input the problem description in its simplest and most intuitive formulation. In particular, the user interface and the required inputs are analogous to one of *Gurobi* [30], a well-consolidated optimization package for conventional solvers.
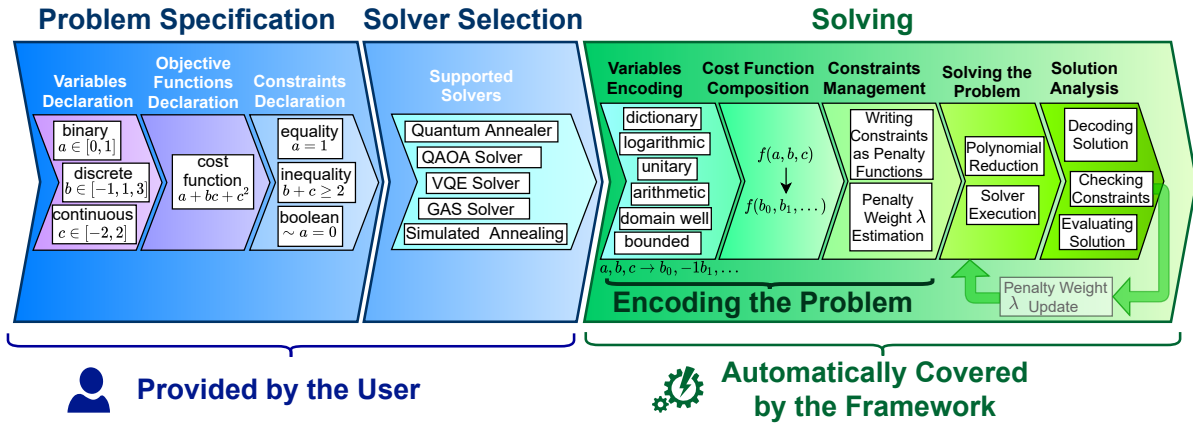
Fig. 1: Quantum Optimization Flow.

The proposed framework aims to assist non-experts in exploring the potential of the quantum solver in the same fashion as classical alternatives. It enables the automated translation of a problem from a classical description to a format compatible with the chosen quantum solver. Moreover, it allows and aids in decoding and evaluating the obtained solution.

Figure 1 summarizes the corresponding steps to be done by the user and those that are automatically covered by the proposed framework. More precisely:

1) The user must provide the *problem specification*, consisting of *variables declaration*, which can be binary (unipolar or bipolar), discrete (specifying the valid values), and continuous (defining the operative range and the wanted precision), *objective functions assertion*, specifying the importance weight, if a multi-objective optimization occurs, as well as *constraints definition*, belonging to equality, inequality, and boolean categories.
2) Then, the user has to *select a solver* among the supported ones, i.e., QA, QAOA, VQE, GAS, or SA;
3) Based on that, the framework *automatically* translates the description into a solver-compatible format and analyzes the obtained solution by executing the following steps.

   a) *Variables encoding*, which consists of describing the multilevel variables as a set of binary ones with one of the techniques described in Section IV-A.
   b) *Composing the cost function* involving only binary variables, as Section IV-B describes.
   c) *Writing the constraints as penalty functions*, and *estimating a proper penalty weight* λ, as described in Sections IV-C and IV-D, respectively.
   d) *Solving the problem* with the chosen optimizer, as described in Section IV-E.
   e) *Solution analysis*, checking the constraint satisfaction, and eventually *updating the penalty weight* λ for obtaining valid outcomes, as described in Sections IV-F and IV-G.

C. Related Works

In recent years, supporting and automating the procedure for describing optimization problems in a quantum-compliant format for quantum solver exploitation has become a focus. Several libraries and some tools have emerged to aid the QUBO formulation process. The main libraries include *pyqubo* [31], [48], *qubovert* [32], *dimod* [33], *Qiskit-optimization* [34], *Fixstarts Amplify* [35] and *openQAOA Entropica* [36], while two frameworks, *AutoQUBO* [37], [49], [50] and QUBO.jl [42], have been proposed in the last two years to meet users' requests for a tool that automates the *entire* procedure. Table I provides a comprehensive overview of all these tools together with a comparison of the features of the framework proposed in this work aims to provide.

More precisely, the table shows that while libraries significantly simplify complex steps in their respective procedures, their main limitation lies in the lack of support for the automatic execution of these steps, restricting their usability to users with at least a minimum level of expertise in the field. Moreover, each library alone does not cover all essential steps completely. On the other hand, the main limitation of AutoQUBO is its lack of support for managing non-binary variables and writing constraints as penalty functions. Furthermore, it is principally designed for *Digital Annealer* (DA) [51]—the quantum-inspired Fujitsu solver—, the SA and the QA, with no support for execution on quantum-circuit-model-based solvers. Finally, QUBO.jl [42] requires as input a JuMP problem (expressed in the Julia language instead of Python, which is the main language for quantum frameworks) and stands out as the first tool supporting the encoding of floating variables. However, it is incomplete concerning the penalties weight assignment.

As can be seen, all existing libraries and frameworks still heavily rely on the end user and do not fully harness the potential for automation. In contrast, to the best of our knowledge, the framework proposed in this work stands out as *the first to guarantee complete coverage of the most crucial steps* of quantum optimization. Moreover, it supports all methods available in the literature for each step, allowing the selection of the most suitable approach for a specific application.

TABLE I: Comparing the support provided by of proposed framework and existing libraries and framework in each step of quantum optimization.

✓ indicates that the corresponding action is performed automatically.

✓ signifies that a proper function is available for implementing the step.

✗ indicates that the method is not fully supported.

[+] denotes that logarithmic encoding is also compatible with bases different from two.

[*] signifies that the encoding techniques can be exploited only for constraints translation.

[†] indicates that the polynomial reduction is implemented by exploiting the corresponding qubovert function.

Supports for each step		Existing Libraries						Existing Frameworks		Proposed
		pyqubo [31]	qubovert [32]	dimod [33]	Qiskit [34]	fixstars [35]	openQAOA [36]	AutoQUBO [37]	QUBO.jl [38]	Framework
Floating Encoding		✗	✗	✗	✗	✗	✗	✗	✓	✓
Integer Encoding	Logarithmic [39]	✓	✓	✓	✓	✓[*]	✗	✗	✓	✓[+]
	Unitary [39]	✓	✓	✗	✗	✓[*]	✗	✗	✓	✓
	Dictionary [39]	✓	✗	✗	✗	✗	✗	✗	✓	✓
	Domain-Wall [40]	✓	✗	✗	✗	✗	✗	✗	✓	✓
	Bounded-Coeff [41]	✗	✗	✗	✗	✗	✗	✗	✓	✓
	Arithmetic [42]	✗	✗	✗	✗	✓[*]	✗	✗	✓	✓
Penalty Functions	Equality [22] [21]	✗	✓	✓	✓	✓	✓	✗	✓	✓
	Inequality [22]	✗	✓	✓	✓	✓	✓	✗	✓	✓
	Boolean [22]	✓	✓	✓	✗	✓	✗	✗	✗	✓
Penalty Weight	UB positive [43]	✗	✗	✗	✗	✗	✗	✗	✗	✓
	MQC [43]	✗	✗	✓	✗	✗	✗	✗	✗	✓
	VLM [44]	✗	✗	✗	✗	✗	✗	✓	✗	✓
	MOMC [43]	✗	✗	✗	✗	✗	✗	✗	✗	✓
	MOC [43]	✗	✗	✗	✗	✗	✗	✗	✗	✓
	UB Naive [45], [46]	✗	✓	✗	✓	✗	✗	✓	✗	✓
	UB posiform [45], [46]	✗	✗	✗	✗	✗	✗	✓	✗	✓
Polynomial Reduction		✓	✓	✓	✗	✓	✗	✓	✓	✓[†]
Solvers	Dwave QA	✓	✗	✓	✗	✓	✗	✓	✓	✓
	QAOA	✗	✗	✗	✓	✗	✓	✗	✓	✓
	VQE	✗	✗	✗	✓	✗	✓	✗	✓	✓
	GAS	✗	✗	✗	✓	✗	✗	✗	✗	✓
	SA	✓	✓	✓	✓	✓	✗	✓	✓	✓
Solution Decoding		✓	✓	✓	✓	✓	✓	✓	✓	✓
Check Constraints		✓	✓	✓	✗	✓	✗	✓	✓	✓
Penalty Update	Sequential [47]	✗	✗	✗	✗	✗	✗	✗	✗	✓
	Scaled [47]	✗	✗	✗	✗	✗	✗	✗	✗	✓
	Binary search [47]	✗	✗	✗	✗	✗	✗	✗	✗	✓

IV. IMPLEMENTATION OF THE PROPOSED FRAMEWORK

This section details the implementation of the proposed framework, designed to automate the optimization of any polynomial cost functions exploiting quantum solvers. Figure 2 shows the workflow considered. The example introduced in the following is examined to adequately describe the steps required for transitioning a generic cost function into a quantum-solver-compliant format.

Example 3. *Let us consider the cost function $a+bc+c^2$, where a is a unipolar binary variable, b is a discrete variable taking values in the set $[-1, 1, 3]$, and c is a continuous variable in the range $[-2, 2]$. The target is to determine the configuration of variables that minimizes the cost function while satisfying the constraint $b + c \geq 2$.*

This abstract example has been chosen because its simplicity enables us to effectively showcase various aspects of the framework. Using this example, the following describes how the steps illustrated in Figure 2 are automatically conducted.

A. Variable Encoding

First, the variables must be converted into a binary format through proper encoding techniques, whose target is to associate with each multilevel variable a set of *binary* ones with *appropriate weights* and, eventually, *offset*. The employed encoding technique depends on the type of variable declared by the user. In particular, for *unipolar binary* variables, a one-to-one association is established without the need for weights or offsets; for *bipolar binary* variables, a one-to-one association is considered with a weight of two and an offset of minus one; for *discrete* variables, a *dictionary* encoding [39] mechanism can be implemented by associating a binary variable with each possible value, ensuring that only one of them can be active in a valid solution; for *continuous* variables, various encoding techniques, for a wanted preci-

Fig. 2: Workflow of the proposed framework. The letters correspond to the subsections where each step is discussed.

sion, are supported such as *dictionary* encoding [39] (as in the discrete case exploiting operative range discretization), *logarithmic* encoding [39], *unitary* encoding [39], *arithmetic progression* encoding [42], *domain-well* encoding [40], and *bounded coefficient* encoding [41].

In the proposed framework, an effort was made to extend the mentioned encoding techniques, initially designed for *integers*, to *real* numbers, even managing an asymmetric operational range. Moreover, the framework allows the declaration of multidimensional arrays of variables. This feature enables a more concise definition of objective functions, allowing the direct expression of matrix-based formulations.

Example 3. *The presented variables can be encoded in the following way:*

- $a \rightarrow b_0$;
- $b \rightarrow -1b_1 + 1b_2 + 3b_3$, *imposing* $b_1 + b_2 + b_3 = 1$;
- $c \rightarrow$:

  - *dictionary considering a precision of 0.5:* $-2b_4 - 1.5b_5 - 1b_6 - 0.5b_7 + 0b_8 + 0.5b_9 + 1b_{10} + 1.5b_{11} + 2b_{12}$, *imposing* $b_4 + b_5 + b_6 + b_7 + b_8 + b_9 + b_{10} + b_{11} + b_{12} = 1$;
  - *logarithmic considering a precision of 0.25 and the base 2:* $0.25b_4 + 0.5b_5 + 1b_6 + 2b_7 + 0.25b_8 - 2$;
  - *unitary considering a precision of 0.5:* $0.5b_4 + 0.5b_5 + 0.5b_6 + 0.5b_7 + 0.5b_8 + 0.5b_9 + 0.5b_{1}0 + 0.5b_{11} - 2$;
  - *arithmetic progression considering a precision of 0.2:* $0.2b_4 + 0.4b_5 + 0.6b_6 + 0.8b_7 + 1b_8 + 1b_9 - 2$;
  - *domain-well considering a precision of 0.5:* $0.5b_4 + 1b_5 + 1.5b_6 + 2b_7 + 2.5b_8 + 2b_9 - 2$ *imposing* $b_i \geq b_{i-1}$;
  - *bounded coefficient considering a precision of 0.5, with bound 1:* $0.5b_4 + 1b_5 + 1b_6 + 1b_7 - 2$.

B. Cost Function Composition

Afterwards, the declared objective functions have to be combined, considering the *aggregation weight*, the optimization sense (adjusting the sign for maximization contributions), and replacing the original variables with the binary ones, as in the following:

$$(\min, f(x), \delta), (\max, g(y), \eta) \rightarrow q(\mathbf{b}) = \delta f(\mathbf{b}) - \eta g(\mathbf{b}), \quad (3)$$

where x and y are the variables declared by the user, $\mathbf{b}$ the array of binary variables for the encoding, $\min$ and $\max$ indicate minimization and maximization, respectively, δ and η are the user preference weights. The result is a cost function considering all the optimization figures of merit involving exclusively binary variables.

Example 3. *Considering the logarithmic encoding mechanism with a precision of 0.25 and the base 2 for variable c, the declared objective function* $a + bc + c^2$ *becomes:*

$$a + bc + c^2 \rightarrow (b_0) + (-1b_1 + 1b_2 + 3b_3)(0.25b_4 + $$
$$+ 0.5b_5 + 1b_6 + 2b_7 + 0.25b_8 - 2) + (0.25b_4 + 0.5b_5 + $$
$$+ 1b_6 + 2b_7 + 0.25b_8 - 2)^2$$

Finally, the expression is expanded.

C. Writing Constraints as Penalty Functions

Subsequently, the problem constraints are transformed into appropriate penalty functions within the framework. This step consists of identifying the type of constraint, replacing the original variable with binary counterparts in the constraint expression, and composing the proper penalty function associated with the constraint, eventually inserting auxiliary variables.

The framework can handle *equality constraints* [22], *inequality constraints* (greater than, greater equal, less than, and

less equal)—expressed as equality constraints with the introduction of auxiliary variables [22], whose necessary precision can be defined by the user or inferred by the expression—, and *boolean constraints* (not, and, or, xor). These constraint categories aim to ensure comprehensive coverage of prevalent situations encountered in real-world scenarios.

Example 3. *The first constraint to evaluate in the proposed example is one deriving from the encoding of variable b, i.e., $b_1 + b_2 + b_3 = 1$. Therefore, the proper penalty function expressing this constraint is $g(\boldsymbol{b}) = (b_1 + b_2 + b_3 - 1)^2$. In this case, the constraint is already written with the final problem binary variables, not requiring the substitution step. The expression can be expanded as:*

$$g(\boldsymbol{b}) = -b_1 - b_2 - b_3 + 2b_1b_2 + 2b_1b_3 + 2b_2b_3 + 1\,,$$

remembering that $b_i = b_i^2$. It is possible to notice, that, as desired, $g(x)$ is null only when exactly one among the variables is set to one and assumes a positive value otherwise. The second constraint is imposed by the user and requires that $b + c \geq 2$. First of all, this has to be written as an equality constraint through the introduction of a real auxiliary variable $\_\_aux$, which should have the operative range $[-(\max(b+c) - 2), 0]$, i.e., $[-3, 0]$. In this way, the constraint becomes $b + c + \_\_aux = 2$, which can be written as $k(b, c) = (b + c + \_\_aux - 2)^2$. It is possible to notice that, choosing proper values of the auxiliary variable, all the b, c configurations satisfying the constraint are not penalized (e.g., for $b = 3$, $c = 1$, the penalty function is null for $\_\_aux = -2$). Then, the variables are replaced in the function with their binary encoding (logarithm is considered for the auxiliary variable) as follows:

$$k(\boldsymbol{b}) = (0.25\_a_0 + 0.5\_a_1 + 1\_a_2 + 1.25\_a_3 - 1b_1 + 1b_2 + 3b_3 + 0.25b_4 + 0.5b_5 + 1b_6 + 2b_7 + 0.25b_8 - 2 - 3)^2\,.$$

Finally, the expression is expanded.

D. Penalty Weight Estimation

Determining the *penalty weight*, denoted as λ, is critical for *effectively penalizing non-valid solutions* while integrating penalty functions with the objective function. In particular, it is necessary to select a λ satisfying the following relation:

$$f(x*) < f(x) + \lambda g(x) \quad \forall x \in S \rightarrow \lambda > \max_{x \in S}\left(\frac{f(x*) - f(x)}{g(x)}\right)$$

where $f(x)$ is the cost function to optimize, $f(x*)$ is the cost function value for the optimal solution $x*$, $g(x)$ is the penalty function and S is the space of infeasible solutions. Accordingly, its evaluation would require knowing the range of values that cost and penalty functions assume (functions bounds). Therefore, *limited complexity methods* for estimating a λ satisfying the relation, i.e., ensuring the correct penalization of unfeasible solutions without compromising the quality of the solution space exploration. The state-of-the-art suggests taking λ equal to a certain percentage of the objective function range, i.e., about *75%-150%* [21].

The framework supports the following methods of literature:

- *Upper bound of objective functions involving only positive coefficients* (UB positive, [43]), which is valid for QUBO problems with all positive coefficients and estimates the objective function upper bound by summing all the QUBO coefficients.
- *Maximum QUBO coefficient* (MQC), which is originally thought for travelling salesman problem and chooses as λ the QUBO coefficient presenting the highest value.
- *Verma and Lewis* (VLM, [44]), which estimates the amount of gain/loss that an objective function can achieve by flipping a single variable. λ is its maximum value.
- *Maximum change in Objective function divided by Minimum Constraint function of infeasible solutions* (MOMC, [43]), which is a modified version of VLM, dividing its λ, by the minimum variation in the constraint function.
- *Maximum value derived from dividing each change in Objective function with the corresponding change in Constraint function* (MOC, [43]), which is derived from VLM, dividing each change in the objective function by the corresponding change in the constraint function and choosing the maximum ratio.
- *Upper bound Naive* (UB Naive, [45], [46]), which estimates the cost function upper bound as the sum of all positive QUBO coefficients and the lower bound as the sum of negatives. The λ is the difference between the upper and lower bound.
- *Upper Bound posiform* (UB posiform, [45], [46]), which computes upper and lower bounds by exploiting the concept of posiform and negaform.

It is possible to notice that MOMC and MOC are unique policies considering not only the objective functions but also the penalties to apply.

In the proposed framework, the values found with the presented method are multiplied by an amount, close to one, which can be chosen by the user, which is different for hard and weak constraints. This provides an additional degree of freedom for the user to adapt the approaches to their needs. Alternatively, the λ's values can be set manually, although is not recommended.

Example 3. *For the considered example, the values found for λ are as follows:*

- *UB positive: Not applicable;*
- *MQC: $\lambda = 6 + 4 = 10$, where 6 is the maximum coefficient and 4 the function offset, for both constraints;*
- *VLM: $\lambda = 12$, for both constraints;*
- *MOMC: $\lambda_0 = 6.19$ and $\lambda_1 = 12$, for the inequality and equality constraints, respectively;*
- *MOC: $\lambda_0 = 1$ and $\lambda_1 = 6$, for the inequality and equality constraints, respectively;*
- *UB Naive: $\lambda = 52.25$, for both constraints*
- *UB posiform: $\lambda = 31.625$, for both constraints.*

E. Solving the Problem

Next, the problem is optimized by exploiting the selected solver. However, a preliminary step involves *reducing polynomials to second order* before proceeding. In our framework, this task is currently performed by functions within the qubovert library. This choice was made for simplicity, as the method for this reduction is essentially standard, with few degrees of freedom and possible optimizations with respect to the other steps of the tool, and was already well-implemented in the library. Unfortunately, polynomial reduction is expensive from the problem complexity point of view, as it involves adding *auxiliary binary variables* to represent polynomials with degrees higher than two.

The framework offers a *common interface* for all supported solvers, with solutions provided in the same format, as discussed in the next section. However, setting optimizer parameters differs between solvers, as they are specific to each solver's requirements.

F. Solution Analysis

The solvers provide in output a solution object comprising:
- the list of obtained *solutions* in binary variables;
- the list of obtained *solutions* in the originally declared variables through a proper *decoding* mechanism;
- list of *energies* found;
- the *best solution* in binary and originally declared variables, i.e., ones corresponding to the lowest energy;
- the *best energy* value.

Moreover, the framework provides a set of methods for analyzing the obtained results:
- Methods for *checking constraint satisfaction* of the best solution or all the obtained solutions, eventually considering weak constraints if a flag is set.
- Method for evaluating the *value* obtained with the best solution *for each objective function* declared (particularly useful in case of multi-objective optimization).
- Method for showing the *cumulative distribution* of the obtained results. To understand the meaning of this, one rule has to be considered: the probability of obtaining the optimal value (or a value close to it) with a solver is higher when its corresponding cumulative distribution is more concentrated on the left of the plot, where the lowest values are located.
- Method for evaluating the *rate of valid solutions* found.
- Method for evaluating the p_{range}, which is the probability of obtaining a final energy lower than a certain value (val_ref) and can be computed as follows:

$$p_{\text{range}} \triangleq \frac{n_{\text{in\_range}}}{n_{\text{tot}}} 100 \,, \tag{4}$$

where $n_{\text{in\_range}}$ is the number of times in which the solver achieved final energy lower than val_ref, and n_{tot} is the number of runs.
- Method for computing the *Time-To-Solution* (TTS), if the proper flag for saving the solver execution time was set. It is a figure of merit commonly employed for comparing

quantum and quantum-inspired approaches with classical solutions [52]–[56]. It is defined as the time required to find a *target solution*, which is the optimal one or a suboptimal with final energy lower than a certain value with a percentage of confidence p_{conf} and can be computed as:

$$\text{TTS} = t_f \frac{\log\left(1 - p_{\text{conf}}\right)}{\log\left(1 - p_{\text{range}}(t_f)\right)} \,, \tag{5}$$

where t_f is the algorithm execution time, $p_{\text{range}}(t_f)$ is the probability of finding energy lower than a certain value, executing the algorithm for a time t_f.
- Method for saving all the information related to the solution in a *JavaScript Object Notation* (JSON) file, allowing the post-processing or the exploitation of the solution at a later time.

Example 3. *In this case, the expected optimal solution in binary variables is:*

$$\{b_0 : 0, b_4 : 1, b_1 : 0, b_5 : 0, b_6 : 1, b_7 : 1, b_8 : 0, b_2 : 0,$$
$$b_3 : 0, \_a_0 : 0, \_a_1 : 0, \_a_2 : 0, \_a_3 : 0, \_a_4 : 0.0\}$$

while written with the original variables is

$$\{a : 0.0, b : 3.0, c : -1.0\}$$

The best energy is equal to -2.0.
It is possible to notice that the solution satisfies both constraints:

$$b_1 + b_2 + b_4 == 1 \rightarrow 1 = 1 \,,$$

and

$$b + c \geq 2 \rightarrow 2.0 \geq 2.0 \,.$$

G. Penalty Weight Update

Finally, the framework provides further instruments, i.e., the possibility of automatically performing the constraints satisfaction check on the best solution and, if the solution is not valid, executing the algorithm again, *adjusting* the value of λ until a valid solution is found or the maximum number of trials t, selected by the user, is exceeded. This could be necessary, even if the tool provides methods for λ estimation, since they are not exact methods, and consequently, their effectiveness could be problem-dependent, thus requiring adjustment.

The framework supports three methods, presented in [47], for this purpose:
- *sequential scaling* (sequential), which multiplies the old λ for a factor equal to 10, increasing it by one order of magnitude;
- *scaled-sequential scaling* (scaled), which increases the λ with the following equation:

$$\lambda_{\text{new}} = \text{round}(\lambda \lambda_{\text{max}}^{\frac{1}{t-1}}) \,,$$

where λ_{max} is the upper bound for the penalty weight;
- *binary-search scaling* (binary search), which increases the λ with the following equation:

$$\lambda_{\text{new}} = \text{round}(\sqrt{\lambda \lambda_{\text{max}}}) \,.$$

Example 3. *Finally, the problem specification of the considered example can be provided to the framework with the following code.*

```
var = Variables()
a = var.add_binary_variable("a")
b = var.add_discrete_variable("b", [-1, 1, 3])
c = var.add_continuous_variable("c", -2, 2, 0.25)
obj_func = ObjectiveFunction()
obj_func.add_objective_function( a + b * c + c**2)
cst = Constraints()
cst.add_constraint("b + c >= 2", variable_precision=True)
prb = Problem()
prb.create_problem(var, cst, obj_func)
```

To solve it with, for example, the Dwave QA, it is sufficient to write the command in the following. The framework automatically performs problem translation and the solver execution, providing the solution object previously described as output.

```
solution = Solver().solve_dwave_quantum_annealer(prb,token=
    token)
```

V. USE CASES

This section summarizes the results obtained from the use cases, evaluating the proposed framework from the user's perspective in terms of effectiveness and flexibility. To this end, the case studies already covered above have been considered: the knapsack problem (chosen to showcase the handling of inequality constraints) and a linear regression task (highlighting the utilization of real variables). The problems can be solved in both instances without requiring expertise in quantum-compliant formulation and with minimal familiarity with quantum solvers.

All details of these case studies, as well as an open-source implementation of the proposed framework, are publicly available on GitHub (https://github.com/cda-tum/mqt-qao).

A. Knapsack

As previously mentioned, the *knapsack* problem [57] aims to define for a set of N_{obj} objects, characterized by weights w_{arr_i} and of preference scores p_{arr_i}, the best subset to put into a bag, ensuring that the total weight does not exceed the maximum W_{max}. The problem is conventionally formulated by associating a variable for each object (obj_i), assuming 1 if the object is in the subset and 0 otherwise, and optimizing the objective function

$$\text{maximize} \quad f(\text{obj}) = \sum_i^{N_{obj}} p_{arr_i} \text{obj}_i = \text{obj}^T p_{arr} \,,$$

subject to

$$\sum_i^{N_{obj}} w_{arr_i} \text{obj}_i = \text{obj}^T w_{arr} \leq W_{max} \,. \quad (6)$$

Traditionally, writing the problem according to QUBO formalism usually requires following the methodology outlined in [58], consisting of the steps reviewed in Section II. However, leveraging the proposed framework substantially streamlines the process. In fact, only the following code is required to provide the problem specification:

```
var = Variables()
obj = vars.add_binary_variables_array("obj", [N_obj])
obj_func = ObjectiveFunction()
obj_func.add_objective_function( np.dot(np.transpose(obj),
    p_arr), minimization=False)
cst = Constraints()
cst.add_constraint(str(np.dot(np.transpose(obj), w_arr)) +
    " <= " + format(W_max))
prb = Problem()
prb.create_problem(var, cst, obj_func)
```

Afterwards, the problem can be solved with the QAOA optimizer (as one possible representative solver) by simply executing the following command:

```
solution = Solver().solve_qaoa_qubo(prb)
```

For providing examples of the results analysis that can be conducted with the framework, Figures 3a and 3b were generated by considering the f3_l-d_kp_4_20 knapsack problem of the 0/1 set, which comprises the weights and preference scores for different sets of objects. In particular, Figure 3a shows the cumulative distribution derived from executing each supported solver one hundred times on the problem, while Figure 3b illustrates the rate of valid solutions, i.e., those satisfying the constraint, and the p_{range} values computed by the framework using Equation 4, with a reference value of -30, proximate to the expected optimal value of -35.

B. Linear Regression

Linear regression [1] is a statistical technique for modeling the relation between variables and others exploitable in a wide range of applications, including machine learning. As discussed above, it consists of identifying the straight line that best fits some available data. The model to optimize can be expressed as

$$\text{minimize} \quad E(w) = \|Xw + Y\| \,,$$

where $X \in \mathbb{R}^{N \times (d+1)}$ is the augmented regression data matrix, N the number of data points in the training set, d the number of features, $Y \in \mathbb{R}^N$ is the regression label of the training data, $w \in \mathbb{R}^{d+1}$ the regression weights and $E(w)$ the Euclidian error function. This can be rewritten in a linear form as

$$\text{minimize} \quad E(w) = w^T X^T X^T X w - 2w^T X^T Y + Y^T Y \,,$$

allowing the exploitation of the framework for describing the problem—establishing the operative range (minv, maxv, precision) of the regression weights—in a practical way, as shown in the following, instead of applying the complex procedure presented in [1], consisting of the steps discussed in Section II.

More precisely, the problem specification can be provided to the framework with the following instructions:

```
var = Variables()
w = var.add_continuous_variables_array("w", [d + 1], minv,
    maxv, precision)
obj_func = ObjectiveFunction()
obj_func.add_objective_function(np.dot(np.dot(np.dot(np.
    transpose(w), np.transpose(X_training)), X_training), w
    ) - 2 * np.dot(np.dot(np.transpose(w), np.transpose(
    X_training)), Y_training)+ np.dot(np.transpose(
    Y_training), Y_training),)
cst = Constraints()
prb = Problem()
prb.create_problem(var, cst, obj_func)
```

Afterwards, the VQE optimizer (as one possible representative solver) can solve it by launching the following command:

```
solution =  Solver().solve_vqe_qubo(prb)
```

For showing the framework tools for results analysis, Figure 3c shows the cumulative distribution obtained by executing one hundred times each supported solver considering two features of the Iris dataset and w range $[-0.25, 0.25]$ with precision 0.25.
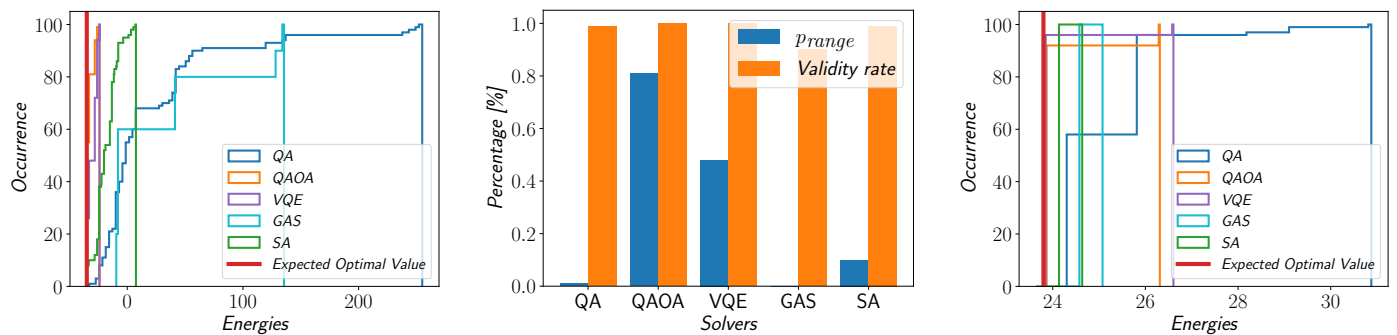
(a) Cumulative distribution of the f3_l-d_kp_4_20 knapsack problem of the 0/1 set, considering VLM method for λ estimation

(b) Rate of valid solution and p_{range} with reference value -30 (Equation 4) for the f3_l-d_kp_4_20 knapsack problem of the 0/1 set, considering an expected optimal energy of -35

(c) Cumulative distribution of linear regression performed considering the Iris dataset, considering two features and w range $[-0.25, 0.25]$, using a precision of 0.25

Fig. 3: Analysis of the results obtained by solving with each supported optimizer the use case problems.

C. Discussion

The two real-world use cases discussed demonstrate the framework's comprehensiveness, adeptly managing constraints and real variables. Observing the code implementations, it becomes evident that the user interface is intuitive and closely resembles conventional optimization packages, enabling users to express their problems in a familiar and conventional way based on variables, constraints, etc. A single black box function completely hides the complexities of translating problems into QUBO formalism and harnessing quantum solvers from non-expert users. This function produces a solution object common to all solvers as output.

For users possessing a certain level of expertise, the framework offers the flexibility to manually specify various aspects, such as the encoding method or the mechanism for λ estimation, as well as the solver parameters, rather than relying on default settings. Furthermore, as shown in Figure 3, the framework provides valuable tools for analyzing the obtained solution and facilitating comparisons between different solvers.

VI. CONCLUSIONS

This work introduced a framework, publicly available on GitHub (https://github.com/cda-tum/mqt-qao) as part of the Munich Quantum Toolkit (MQT), designed to empower non-experts in quantum computing and QUBO formulation to explore the potential of quantum solutions for optimization problems. To this end, we first reviewed the procedure required for solving an optimization problem with a quantum solver. Then, we demonstrated how the framework automates this process, providing users with an interface as similar as possible to one of the commonly used packages for conventional optimization. This automation conceals the complexity of formulating problems according to the QUBO formalism and managing quantum solvers. The effectiveness and flexibility of the framework have been shown in two different case studies: the knapsack problem and linear regression modeling. In both cases, it could be seen that, after the declaration of the problem specifications, a solution can be obtained by executing a simple black-box command.

Even though the considered use case has shown the potential and the resulting benefit of the framework for the users, opportunities for enhancement and expansion remain. First, partial or total automation of quantum solver parameter adaptation tailored to specific problems could enhance efficiency. Moreover, the framework can be exploited for performing comparisons among quantum solvers and developing a prediction mechanism to identify the most suitable solver for a given problem, automating the solver selection. Finally, recognizing that the main limitation of the tool is the requirement of providing input cost functions in a polynomial form— excluding non-linearities such as exponential or trigonometric functions—a linearization step can be added, thus expanding its application domain.

In conclusion, the proposed framework lays the foundation for automating quantum computing solutions and broadening access to quantum optimization for a wider range of users.

REFERENCES

[1] P. Date, D. Arthur, and L. Pusey-Nazzaro, "Qubo formulations for training machine learning models," *Scientific reports*, vol. 11, no. 1, p. 10029, 2021. https://doi.org/10.1038/s41598-021-89461-4.

[2] F. Hu, B.-N. Wang, N. Wang, and C. Wang, "Quantum machine learning with d-wave quantum computer," *Quantum Engineering*, vol. 1, no. 2, p. e12, 2019. https://doi.org/10.1002/que2.12.

[3] J. Zhang, G. L. Bianco, and J. C. Beck, "Solving job-shop scheduling problems with qubo-based specialized hardware," in *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 32, pp. 404–412, 2022. https://doi.org/10.1609/icaps.v32i1.19826.

[4] T. Ohyama, Y. Kawamoto, and N. Kato, "Resource allocation optimization by quantum computing for shared use of standalone irs," *IEEE Transactions on Emerging Topics in Computing*, 2023. https://doi.org/10.1109/TETC.2023.3292355.

[5] R. Wille, L. Berent, T. Forster, J. Kunasaikaran, K. Mato, T. Peham, N. Quetschlich, D. Rovara, A. Sander, L. Schmid, D. Schoenberger, Y. Stade, and L. Burgholzer, "The MQT Handbook: A Summary of Design Automation Tools and Software for Quantum Computing," in *International Conference on Quantum Software (QSW)*, 2024. A live version of this document is available at https://mqt.readthedocs.io.

[6] T. Kadowaki and H. Nishimori, "Quantum annealing in the transverse ising model," *Physical Review E*, vol. 58, no. 5, p. 5355, 1998. https://doi.org/10.1103/PhysRevE.58.5355.

[7] B. K. Chakrabarti, H. Leschke, P. Ray, T. Shirai, and S. Tanaka, "Quantum annealing and computation: challenges and perspectives," *Philosophical Transactions of the Royal Society A*, vol. 381, no. 2241, p. 20210419, 2023. https://doi.org/10.1098/rsta.2021.0419.

[8] A. Rajak, S. Suzuki, A. Dutta, and B. K. Chakrabarti, "Quantum annealing: an overview," *Philosophical Transactions of the Royal Society A*, vol. 381, no. 2241, p. 20210417, 2023. https://doi.org/10.1098/rsta.2021.0417.

[9] D. Venturelli, S. Mandrà, S. Knysh, B. O'Gorman, R. Biswas, and V. Smelyanskiy, "Quantum optimization of fully connected spin glasses," *Physical Review X*, vol. 5, no. 3, p. 031040, 2015. https://doi.org/10.1103/PhysRevX.5.031040.

[10] M. W. Johnson, M. H. Amin, S. Gildert, T. Lanting, F. Hamze, N. Dickson, R. Harris, A. J. Berkley, J. Johansson, P. Bunyk, *et al.*, "Quantum annealing with manufactured spins," *Nature*, vol. 473, no. 7346, pp. 194–198, 2011. https://doi.org/10.1038/nature10012.

[11] M. A. Nielsen and I. L. Chuang, *Quantum computation and quantum information*. Cambridge ; New York: Cambridge University Press, 10th anniversary ed ed., 2010.

[12] K. Blekos, D. Brand, A. Ceschini, C.-H. Chou, R.-H. Li, K. Pandya, and A. Summer, "A review on quantum approximate optimization algorithm and its variants," *arXiv preprint arXiv:2306.09198*, 2023. https://doi.org/10.48550/arXiv.2306.09198.

[13] E. Farhi, J. Goldstone, and S. Gutmann, "A quantum approximate optimization algorithm," *arXiv preprint arXiv:1411.4028*, 2014. https://doi.org/10.48550/arXiv.1411.4028.

[14] J. Tilly, H. Chen, S. Cao, D. Picozzi, K. Setia, Y. Li, E. Grant, L. Wossnig, I. Rungger, G. H. Booth, *et al.*, "The variational quantum eigensolver: a review of methods and best practices," *Physics Reports*, vol. 986, pp. 1–128, 2022. https://doi.org/10.1016/j.physrep.2022.08.003.

[15] A. Peruzzo, J. McClean, P. Shadbolt, M.-H. Yung, X.-Q. Zhou, P. J. Love, A. Aspuru-Guzik, and J. L. O'brien, "A variational eigenvalue solver on a photonic quantum processor," *Nature communications*, vol. 5, no. 1, p. 4213, 2014. https://doi.org/10.1038/ncomms5213.

[16] D. Bulger, W. P. Baritompa, and G. R. Wood, "Implementing pure adaptive search with grover's quantum algorithm," *Journal of optimization theory and applications*, vol. 116, pp. 517–529, 2003. https://doi.org/10.1023/A:1023061218864.

[17] A. Gilliam, S. Woerner, and C. Gonciulea, "Grover adaptive search for constrained polynomial binary optimization," *Quantum*, vol. 5, p. 428, 2021. https://doi.org/10.22331/q-2021-04-08-428.

[18] Y. Sano, K. Mitarai, N. Yamamoto, and N. Ishikawa, "Accelerating grover adaptive search: Qubit and gate count reduction strategies with higher-order formulations," *arXiv preprint arXiv:2308.01572*, 2023. https://doi.org/10.48550/arXiv.2308.01572.

[19] Y. Sano, M. Norimoto, and N. Ishikawa, "Qubit reduction and quantum speedup for wireless channel assignment problem," *IEEE Transactions on Quantum Engineering*, 2023. https://doi.org/10.1109/TQE.2023.3293452.

[20] L. Giuffrida, D. Volpe, G. A. Cirillo, M. Zamboni, and G. Turvani, "Engineering grover adaptive search: Exploring the degrees of freedom for efficient qubo solving," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 12, no. 3, pp. 614–623, 2022. https://doi.org/10.1109/JETCAS.2022.3202566.

[21] F. Glover, G. Kochenberger, and Y. Du, "A tutorial on formulating and using qubo models," *arXiv preprint arXiv:1811.11538*, 2018. https://doi.org/10.48550/arXiv.1811.11538.

[22] E. Combarro, S. Gonzalez-Castillo, and A. Di Meglio, "A practical guide to quantum machine learning and quantum optimization: Hands-on approach to modern quantum algorithms," 2023.

[23] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, pp. 671–680, May 1983. https://doi.org/10.1126/science.220.4598.671.

[24] M. Fingerhuth, T. Babej, and P. Wittek, "Open source software in quantum computing," *PloS one*, vol. 13, no. 12, p. e0208561, 2018. https://doi.org/10.1371/journal.pone.0208561.

[25] G. G. Guerreschi, J. Hogaboam, F. Baruffa, and N. P. Sawaya, "Intel quantum simulator: A cloud-ready high-performance simulator of quantum circuits," *Quantum Science and Technology*, vol. 5, no. 3, p. 034007, 2020. https://doi.org/10.1088/2058-9565/ab8505.

[26] A. Zulehner and R. Wille, "Advanced simulation of quantum computations," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 5, pp. 848–859, 2018. https://doi.org/10.1109/TCAD.2018.2834427.

[27] S. Hillmich, I. L. Markov, and R. Wille, "Just like the real thing: Fast weak simulation of quantum computation," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*, pp. 1–6, IEEE, 2020. https://doi.org/10.1109/DAC18072.2020.9218555.

[28] T. Vincent, L. J. O'Riordan, M. Andrenkov, J. Brown, N. Killoran, H. Qi, and I. Dhand, "Jet: Fast quantum circuit simulations with parallel task-based tensor-network contraction," *Quantum*, vol. 6, p. 709, 2022. https://doi.org/10.22331/q-2022-05-09-709.

[29] B. Villalonga, S. Boixo, B. Nelson, C. Henze, E. Rieffel, R. Biswas, and S. Mandrà, "A flexible high-performance simulator for verifying and benchmarking quantum circuits implemented on real hardware," *npj Quantum Information*, vol. 5, no. 1, p. 86, 2019. https://doi.org/10.1038/s41534-019-0196-1.

[30] gurobipy documentation, "gurobipy documentation." https://pypi.org/project/gurobipy/, Accessed: December 15, 2023.

[31] P. Documentation, "Pyqubo documentation - getting started." https://pyqubo.readthedocs.io/en/latest/getting_started.html, Accessed: December 15, 2023.

[32] qubovert Documentation, "qubovert documentation - getting started." https://qubovert.readthedocs.io/en/latest/, Accessed: December 15, 2023.

[33] dimod Documentation, "dimod documentation - getting started." https://docs.ocean.dwavesys.com/en/stable/docs_dimod/, Accessed: December 15, 2023.

[34] qiskit-optimization Documentation, "qiskit-optimization documentation - getting started." https://qiskit.org/ecosystem/optimization/index.html, Accessed: December 15, 2023.

[35] fixstarts Documentation, "fixstarts documentation - getting started." https://amplify.fixstars.com/en/docs/, Accessed: December 15, 2023.

[36] openQAOA entropica labs Documentation, "openqaoa entropica labs documentation - getting started." https://el-openqaoa.readthedocs.io/en/latest/, Accessed: December 15, 2023.

[37] autoqubo GitHub repository, "autoqubo github repository." https://github.com/FujitsuResearch/autoqubo, Accessed: December 15, 2023.

[38] Q. G. repository, "Qubo.jl github repository." https://github.com/psrenergy/QUBO.jl, Accessed: December 15, 2023.

[39] K. Tamura, T. Shirai, H. Katsura, S. Tanaka, and N. Togawa, "Performance comparison of typical binary-integer encodings in an ising machine," *IEEE Access*, vol. 9, pp. 81032–81039, 2021. https://doi.org/10.1109/ACCESS.2021.3081685.

[40] N. Chancellor, "Domain wall encoding of discrete variables for quantum annealing and qaoa," *Quantum Science and Technology*, vol. 4, no. 4, p. 045004, 2019. https://doi.org/10.1088/2058-9565/ab33c2.

[41] S. Karimi and P. Ronagh, "Practical integer-to-binary mapping for quantum annealers," *Quantum Information Processing*, vol. 18, no. 4, p. 94, 2019. https://doi.org/10.1007/s11128-019-2213-x.

[42] P. M. Xavier, P. Ripper, T. Andrade, J. D. Garcia, N. Maculan, and D. E. B. Neira, "Qubo.jl: A julia ecosystem for quadratic unconstrained binary optimization," *arXiv preprint arXiv:2307.02577*, 2023. https://doi.org/10.48550/arXiv.2307.02577.

[43] M. Ayodele, "Penalty weights in qubo formulations: Permutation problems," in *European Conference on Evolutionary Computation in Combinatorial Optimization (Part of EvoStar)*, pp. 159–174, Springer, 2022. https://doi.org/10.1007/978-3-031-04148-8_11.

[44] A. Verma and M. Lewis, "Penalty and partitioning techniques to improve performance of qubo solvers," *Discrete Optimization*, vol. 44, p. 100594, 2022. https://doi.org/10.1016/j.disopt.2020.100594.

[45] E. Boros and P. L. Hammer, "Pseudo-boolean optimization," *Discrete applied mathematics*, vol. 123, no. 1-3, pp. 155–225, 2002. https://doi.org/10.1016/S0166-218X(01)00341-9.

[46] E. Boros, P. L. Hammer, and G. Tavares, "Preprocessing of unconstrained quadratic binary optimization," tech. rep., Technical Report RRR 10-2006, RUTCOR, 2006.

[47] M. D. García, M. Ayodele, and A. Moraglio, "Exact and sequential penalty weights in quadratic unconstrained binary optimisation with a digital annealer," in *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pp. 184–187, 2022. https://doi.org/10.1145/3520304.3528925.

[48] M. Zaman, K. Tanahashi, and S. Tanaka, "Pyqubo: Python library for mapping combinatorial optimization problems to qubo form," *IEEE Transactions on Computers*, vol. 71, no. 4, pp. 838–850, 2021. https://doi.org/10.1109/TC.2021.3063618.

[49] A. Moraglio, S. Georgescu, and P. Sadowski, "AutoQubo: data-driven automatic QUBO generation," in *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pp. 2232–2239, 2022. https://doi.org/10.1145/3520304.3533965.

[50] J. Pauckert, M. Ayodele, M. D. García, S. Georgescu, and M. Parizy, "Autoqubo v2: Towards efficient and effective qubo formulations for ising machines," in *Proceedings of the Companion Conference on Genetic and Evolutionary Computation*, pp. 227–230, 2023. https://doi.org/10.1145/3583133.3590662.

[51] M. Aramon, G. Rosenberg, E. Valiante, T. Miyazawa, H. Tamura, and H. G. Katzgraber, "Physics-inspired optimization for quadratic unconstrained problems using a digital annealer," *Frontiers in Physics*, vol. 7, p. 48, 2019. https://doi.org/10.3389/fphy.2019.00048.

[52] I. Hen, J. Job, T. Albash, T. F. Rønnow, M. Troyer, and D. A. Lidar, "Probing for quantum speedup in spin-glass problems with planted solutions," *Physical Review A*, vol. 92, no. 4, p. 042325, 2015. https://doi.org/10.1103/PhysRevA.92.042325.

[53] S. Mandra, Z. Zhu, W. Wang, A. Perdomo-Ortiz, and H. G. Katzgraber, "Strengths and weaknesses of weak-strong cluster problems: A detailed overview of state-of-the-art classical heuristics versus quantum approaches," *Physical Review A*, vol. 94, no. 2, p. 022337, 2016. https://doi.org/10.1103/PhysRevA.94.022337.

[54] T. Albash and D. A. Lidar, "Demonstration of a scaling advantage for a quantum annealer over simulated annealing," *Physical Review X*, vol. 8, no. 3, p. 031016, 2018. https://doi.org/10.1103/PhysRevX.8.031016.

[55] M. Kowalsky, T. Albash, I. Hen, and D. A. Lidar, "3-regular three-xorsat planted solutions benchmark of classical and quantum heuristic optimizers," *Quantum Science and Technology*, vol. 7, no. 2, p. 025008, 2022. https://doi.org/10.1088/2058-9565/ac4d1b.

[56] M. R. Zielewski and H. Takizawa, "A method for reducing time-to-solution in quantum annealing through pausing," in *International Conference on High Performance Computing in Asia-Pacific Region*, pp. 137–145, 2022. https://doi.org/10.1145/3492805.3492815.

[57] H. M. Salkin and C. A. De Kluyver, "The knapsack problem: a survey," *Naval Research Logistics Quarterly*, vol. 22, no. 1, pp. 127–144, 1975. https://doi.org/10.1002/nav.3800220110.

[58] R. A. Quintero and L. F. Zuluaga, "Characterizing and benchmarking qubo reformulations of the knapsack problem," tech. rep., Technical Report. Department of Industrial and Systems Engineering, Lehigh . . . , 2021.