

A Framework to Formulate Pathfinding Problems for Quantum Computing

Damian Rovara*

Nils Quetschlich*

Robert Wille*^{†‡}

*Chair for Design Automation, Technical University of Munich, Germany

[†]Software Competence Center Hagenberg GmbH (SCCH), Austria

[‡]Munich Quantum Software Company GmbH, Garching near Munich, Germany

damian.rovara@tum.de

nils.quetschlich@tum.de

robert.wille@tum.de

<https://www.cda.cit.tum.de/research/quantum>

Abstract—With the applications of quantum computing becoming more and more widespread, finding ways that allow end users without experience in the field to apply quantum computers to solve their individual problems is becoming a crucial task. However, current optimization algorithms require problem instances to be posed in complex formats that are challenging to formulate, even for experts. In particular, the *Quadratic Unconstrained Binary Optimization (QUBO)* formalism employed by many quantum optimization algorithms, such as the *Quantum Approximate Optimization Algorithm (QAOA)*, involves the mathematical rewriting of constraints under strict conditions. To facilitate this process, we propose a framework to *automatically* generate QUBO formulations for pathfinding problems. This framework allows users to translate their specific problem instances into formulations that can be passed directly to quantum algorithms for optimization without requiring any expertise in the field of quantum computing. It supports three different encoding schemes that can easily be compared without requiring manual reformulation efforts. The resulting QUBO formulations are robust and efficient, reducing the previously tedious and error-prone reformulation process to a task that can be completed in a matter of seconds. In addition to an open-source Python package available on <https://github.com/munich-quantum-toolkit/problemsolver>, we also provide a graphical user interface accessible through the web (<https://munich-quantum-toolkit.github.io/problemsolver/>), which can be used to operate the framework without requiring the end user to write any code.

I. INTRODUCTION

The task of finding a path in a directed graph, subject to a set of constraints, is a problem class that has been thoroughly investigated for its complexity and utility. These *pathfinding* problems appear in a large number of application scenarios [1]–[4], and finding exact or approximate solutions to them is a crucial component to solving many problems in the real world.

While efficient classical exact algorithms exist for many pathfinding problems [5], [6], a large number of pathfinding problems, such as the *Traveling Salesperson Problem (TSP)*, the *Sequential Ordering Problem (SOP)*, or the *Multi-Agent Pathfinding Problem*, still lack approaches that provide satisfactory results for real-world problem instances [7]–[9]. Thus, classical methods for approximating the optimal solution to such problems have been an essential area of research for decades.

However, while research on these classical methods has resulted in continual improvements in efficiency and solution quality, alternative approaches also have the potential to complement these methods and even overcome challenges they cannot easily address. Quantum computing solutions, in particular, offer the possibility of revolutionary improvements, as new and rigorously revised algorithms are still regularly proposed for the solution of many optimization problems.

While many such quantum algorithms exploit different optimization approaches, a large number of them are based on the *Quadratic Unconstrained Binary Optimization (QUBO)*, [10], [11] framework. Such algorithms

can tackle individual optimization problems by first formulating them as QUBO problems.

As research in the field of quantum computing is gradually improving with respect to both physical devices and logical algorithms, and their accessibility becomes more and more widespread through service providers such as IBM or Amazon Web Services, interest in the application of quantum algorithms to solve a large variety of different problems is spreading rapidly to an equally diverse range of users. However, while these users may have deep knowledge in their respective domains, they are not necessarily guaranteed to be as experienced in the area of QUBO or quantum computing, and, thus, the development of tools (such as, e.g., proposed in [12]–[16]) that aid end users by shielding them from concepts requiring deeper knowledge of quantum computing has grown in relevance.

Unfortunately, the process of solving individual problems with existing quantum optimization tools is not always straightforward. Input formats for quantum algorithms, such as the QUBO formalism, require extensive mathematical rewriting, a challenging task even for experts already experienced with it. While tools exist that aid the user in this process, they are not able to fully shield them from all the required expertise and still pose significant challenges for many classes of problems (this is discussed in more detail later in Section III).

To solve these problems, this work proposes a novel solution for the *automatic* generation of QUBO formulations of pathfinding problems. We propose a framework that allows end users to define problem instances through the conceptual descriptions of their constraints. These constraints are then individually translated into their corresponding QUBO formulations and finally combined into a single QUBO cost function.

This approach effectively shields users from any required expertise in the field of quantum computing, as the resulting QUBO cost functions can be translated directly into quantum circuits for the solution of individual problems. Furthermore, the framework allows the construction of the result in multiple formats. This allows end users to not only use the generated QUBO formulations in quantum algorithms but also apply them to classical solvers for comparison and evaluation.

Finally, the framework also supports a total of three different encoding schemes that determine how the individual constraints are translated into intermediate quadratic cost functions. While, typically, the evaluation of different encodings requires the QUBO formulation to be remade from scratch, the proposed framework allows users to switch between the different encodings with no additional effort.

II. BACKGROUND

This section provides a short review of the *pathfinding problems* considered throughout this work as well as a general overview on QUBO formulations. Following this, related work in the area is discussed.

A. Pathfinding Problems

Pathfinding problems constitute a problem class that contains a broad range of various graph-based optimization problems with a myriad of applications in the real world, including logistics, scheduling, routing, artificial intelligence, and many more [7]–[9].

More formally, a pathfinding problem is described by a directed graph $G = (V, E)$ with a set of vertices V and a set of edges E . Each edge $(u, v) \in E$ with $u, v \in V$ is assigned a weight $w(u, v) \in \mathbb{R}$. A path π is defined as a sequence of vertices $(v_{i_1}, v_{i_2}, \dots, v_{i_n}) \in V^n$ such that $(v_{i_j}, v_{i_{j+1}}) \in E$ for all $j \in [1, n - 1]$. Based on this, the weight of path π is defined as $w(\pi) = \sum_{(v_{i_j}, v_{i_{j+1}}) \in \pi} w(v_{i_j}, v_{i_{j+1}})$. A pathfinding problem is then given by (1) a set of constraints \mathcal{C} on the graph G , (2) an objective function to determine if the total weight should be minimized or maximized, and (3) the task to find a path that satisfies all constraints and optimizes the objective function.

B. QUBO Formulation

A prominent method for solving a variety of pathfinding problems using quantum computing is to translate them into a *Quadratic Unconstrained Binary Optimization* (QUBO, [11]) problem. This format, used by many frequently used quantum algorithms for optimization problems, allows a large number of different problems to be encoded into a single, optimizable cost function.

QUBO problems are based on a vector of binary variables \mathbf{x} . A cost function $C(\mathbf{x})$ can then be defined in the form $C(\mathbf{x}) = \sum_i c_i x_i + \sum_{i < j} c_{i,j} x_i x_j$ [17]. The optimal solution to a QUBO problem is the vector \mathbf{x}^* that minimizes $C(\mathbf{x}^*)$. Alternatively, the problem can be formulated as a QUBO matrix Q , such that $C(\mathbf{x}) = \mathbf{x}^T Q \mathbf{x}$.

Formulating combinatorial optimization problems in the framework of QUBO problems is an active area of research [17]–[19]. To define a QUBO formulation for a specific problem, one first has to select an adequate *encoding* to represent each potential state of the problem domain as a vector of binary variables \mathbf{x} . Based on such an encoding, a cost function has to be constructed such that the optimal assignment of \mathbf{x} for this cost function corresponds to the optimal solution of the problem.

In particular, each term of such a cost function must be at most quadratic in the use of binary variables. In cases where more than two variables must be multiplied, auxiliary variables can be introduced to keep the term quadratic [20], [21].

Generally, problems defined over several constraints can be transformed into their QUBO cost functions by translating each constraint C_i into $C_i(\mathbf{x})$ individually and, then, constructing $C(\mathbf{x}) = \sum_i P_i C_i(\mathbf{x})$, such that $C(\mathbf{x})$ is minimized if and only if all constraints C_i are satisfied. Here, P_i are penalty factors chosen so that the penalty from any violation of an individual constraint will always dominate over the potential gain by such an assignment for other constraints. Table I shows an example of such cost functions to solve the Traveling Salesperson Problem (TSP), finding the shortest length loop that visits each vertex in a graph exactly once.

Once a QUBO problem is defined, various quantum computing approaches can be used to find its optimal solution, e.g., based on variational quantum algorithms (such as the *Quantum Approximate Optimization Algorithm* (QAOA, [22]), and the *Variational Quantum Eigensolver* (VQE, [23])), *Quantum Annealing* [24], and *Grover Adaptive Search* [25].

TABLE I
COST FUNCTIONS THAT ENCODE THE CONSTRAINTS REQUIRED FOR TSP.

| Constraint | Cost Function $C_i(\mathbf{x})$ |
|-----------------------------------|---|
| (1) $ \pi = V $ | $\sum_{v \in V} \left(1 - \sum_{1 \leq i \leq V } x_{v,i} \right)^2$ |
| (2) $\forall v \in V : v \in \pi$ | |
| (4) $w(\mathbf{x})$ is minimal | $\sum_{(u,v) \in E} \sum_{1 \leq i \leq V } A_{u,v} x_{u,i} x_{v,i+1}$ |

C. Related Work

Due to the complexity of creating QUBO formulations from arbitrary optimization problems, finding alternative approaches to reduce the burden on the end user is an essential area of research, and a wide range of related work exists in this field.

Multiple tools have already been devised that tackle the process of automatic QUBO formulation from different angles. These tools typically allow the user to reformulate optimization problems into a single QUBO cost function or various other output formats that can be solved using classical or quantum algorithms. Representatives are the Python tool *qubovert* [26], IBM's *DOcplex* [27], and Qiskit's [28] *qiskit-optimization* library.

A shared property of all three of these existing tools is that they focus on constructing QUBO formulations from mathematical and logical expressions, i.e., they require the end user to already provide their problem in a numeric format. Since such formulations do not always exist naturally for arbitrary optimization problems, this still requires dedicated expertise from the end user, who is usually a domain expert in their field but not in quantum computing or QUBO. For pathfinding problems, in particular, translating their constraints and corresponding graph structure into this form leads to all the same challenges as creating the QUBO formulation, still posing a significant problem as discussed next.

At the same time, classical approaches to find solutions to pathfinding problems are also being developed. In 2024, *PyVRP* [29] was introduced to provide an efficient and easily usable framework to solve the *Vehicle Routing Problem* (VRP). Similarly, *NVIDIA cuOpt* [30] provides GPU-accelerated solutions for a large number of optimization problems, including the VRP. The *Open Source Routing Machine* [31] is a routing engine that efficiently solves a large number of pathfinding problems on *OpenStreetMap* data.

III. MOTIVATION

This section motivates the methodology proposed in this work by highlighting the challenges of manual QUBO reformulation and devising a general idea to mitigate them.

A. Current Bottlenecks and Challenges

While solving problems given with QUBO formulations is often straight-forward with existing tools and algorithms, the same cannot be said about the construction of the QUBO formulations itself: commonly, optimization problems have to be reformulated into their QUBO representations by the end users, a time-consuming and error-prone task that has to be performed for each new problem to be considered, and which, even with the help of existing software tools that may automate individual steps of the process, suffers from multiple challenges:

- 1) In many cases, the expertise of the end user is limited to the domain of the particular optimization problem to solve rather than the full area of QUBO formulations. Thus, translating individual problems into the QUBO

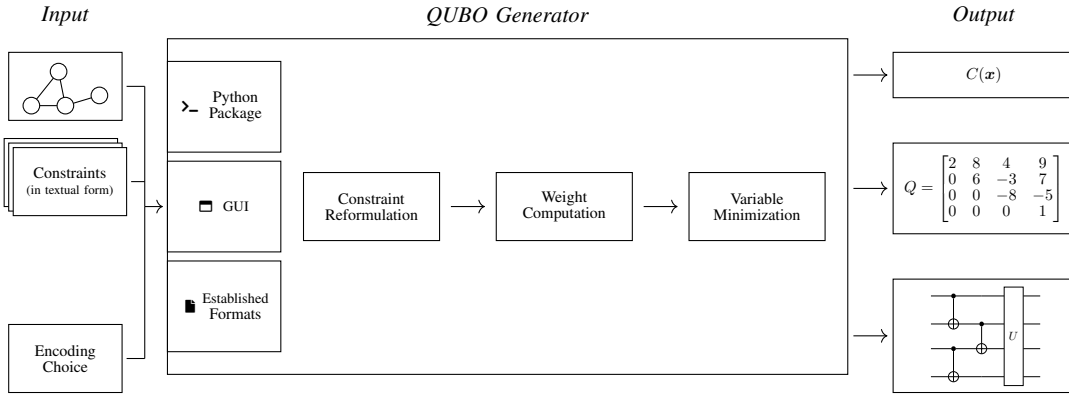


Fig. 1. The workflow of the proposed framework. The user can use a *GUI* to define their desired constraints or use the *Python package* directly. Alternatively, further input formats such as a *JSON* format or the *TSPLib* format are supported. The output, representing a QUBO formulation of the problem, can then be returned using different formats. These formats can be translated automatically into quantum circuits or classical algorithms to solve the problem.

formalism may be difficult or not feasible at all for a large number of users.

- 2) As a math-heavy task, the large number of binary variables required for many problems can easily cause confusion, and the interaction of variables in the cost functions can have unwanted effects. Hence, the entire process is prone to errors that are usually difficult to detect and correct at later stages.
- 3) Only quadratic terms can be considered in the setting of QUBO problems. No term may consist of a product of more than two different variables. Any terms involving three or more binary variables have to be reformulated before being used in a QUBO problem.
- 4) Only unconstrained optimization problems are part of the QUBO framework. Additional constraints must be integrated into the optimization criterion by finding fitting penalty functions as described in Section II.

While tools and frameworks exist that automate the translation from specific optimization problems into QUBO formulations, as reviewed in Section II-C, they are often unable to shield end users entirely from required expertise.

Example 1. Consider the TSP, which aims to find the shortest path that visits a set of vertices exactly once before returning to the starting vertex. Creating a QUBO formulation for it with tools such as qiskit-optimization, reviewed in Section II-C, involves the following steps:

- 1) Create a quadratic programming model with a set of binary variables $e_{i,j}$ that have the value 1 if and only if edge (i, j) is in the solution graph.
- 2) For each vertex i enforce the constraints $\sum_{j \in V} e_{i,j} = 1$ and $\sum_{j \in V} e_{j,i} = 1$.
- 3) For each pair of vertices (i, j) where i is not the starting vertex, enforce the subtour elimination constraint $p_i - p_k + |V| * e_{i,j} \leq |V| - 1$.
- 4) Minimize the expression $\sum_{(i,j) \in E} e_{i,j} w(i, j)$.

This quadratic programming model can then be translated into its QUBO formulation automatically.

Although the method reviewed above allows the user to create models with constraints, it still requires them to overcome the challenges of formulating these models mathematically, ensuring that the constraints are supported by the framework, and forcing the end user to determine a fitting encoding of their own. In this work, we consider a solution to better support the end user in this task.

B. General Idea

The solution proposed in this paper rests on the following general idea: When similar problem instances are considered, the corresponding QUBO formulations often only differ in nuances. For example, for pathfinding problems, the input encoding can often be re-used, and only the encoding of constraints has to be adapted between different problem instances. Furthermore, even these constraints are often shared among various problems (e.g., both the TSP and the SOP consider paths that visit all vertices exactly once), and their QUBO formulation can be leveraged multiple times.

Based on these observations, we propose a framework as sketched in Fig. 1 to provide a fully automated approach to generate QUBO formulations for almost arbitrary pathfinding problems while avoiding the challenges mentioned earlier. In this representation, problem instances can be solved automatically with the help of existing classical tools and quantum algorithms.

More precisely, this framework will provide the following features:

- 1) *Natural Problem Input Interface:* End users should be shielded as much as possible from the tedious and error-prone steps of formulating QUBO problems. Additionally, the pathfinding problem input formulation should be as easy as possible. Therefore, the proposed framework supports a wide range of problem descriptions, including textual descriptions, existing formats, and GUIs, which allow users to provide the graph and the constraints of their considered problems. Furthermore, all input methods allow the user to easily select and switch among three supported encoding types, giving them the freedom to test and compare the different encodings without any additional work.
- 2) *Automated Constraint Translation:* The framework offers an automatic and mathematically correct translation of the given problem into a QUBO formulation, independent of the number of required constraints defining the problem and the arising complexity from their combination. This is encapsulated in the QUBO Generator module sketched in the middle of Fig. 1.
- 3) *Multiple Output Formats:* The resulting QUBO formulation has to be passed to existing classical tools or quantum algorithms to perform the optimization procedure. To this end, the proposed framework supports a large number of output formats. This includes multiple supported output granularities, ranging from a fully prepared quantum circuit to mathematical formulations of cost functions as

visualized on the right of Fig. 1.

Following this approach, the proposed framework allows end users without quantum computing knowledge to generate QUBO formulations automatically for the considered problems with low effort, significantly simplifying the usage of quantum computers for this kind of problems.

IV. PROPOSED FRAMEWORK

In this section, we describe the implementation of the framework sketched above. For more details and for actually trying out the framework, we refer to the open-source implementation on <https://github.com/munich-quantum-toolkit/problemsolver>, implemented as part of the *Munich Quantum Toolkit* (MQT, [32]). To this end, we focus on the core concepts of the *classical input interface*, the *automated translation* process, and the generated *output formats*.

A. Natural Problem Input Interface

This section defines the inputs involved during the workflow of the proposed framework, shown in Fig. 1. As reviewed in Section II, the key components for the construction of a QUBO formulation for pathfinding problems are:

- *Problem Graph*: The graph considered for the individual problem instance.
- *Constraints*: The constraints required to correctly identify the specific problem, in a human-readable form, unrelated to their actual QUBO encodings.
- *Encoding Choice*: The encoding to be used for the representation of the found paths, as a choice among three supported encodings.

As mentioned above, the proposed framework aims to minimize the work required for the user to define and reformulate their specific problems, shielding them from the knowledge required outside their expertise. To this end, it supports multiple interfaces for user interaction (examples are presented later in Section V): As a *Python library*, end users can directly access the framework's core functionality through the specialized classes for QUBO generation. The framework also provides a *graphical user interface* that allows the end user to define constraints and problem settings in a no-code fashion. Finally, the proposed framework also supports established input formats. This allows end users to provide problem definitions in formats they are already familiar with, including a pre-defined *JSON format*, as well as a subset of the established *TSPLib* [33] specification format, which is regularly used to define several pathfinding problems for benchmarking purposes. Such a wide range of input formats drastically reduces entry barriers for users who may not have experience with quantum computing but have worked with pathfinding problems before.

This broad range of input formats allows end users to select the format with which they feel the most comfortable, significantly reducing the hurdle for the creation of QUBO formulations.

B. Automated Constraint Translation

Once problem instance definition has been passed to the framework, it can generate a QUBO formulation for the problem automatically by translating and combining all individual constraints. This process requires a valid mathematical formulation for each combination of constraints and encodings.

The following describes the corresponding translation process. To this end, the particular constraints are explored for a given graph $G = (V, E)$. Further, N represents the maximum path length, Π the set of all paths to be found for the solution of the problem, and $\pi^{(i)}$ the i -th of these paths.

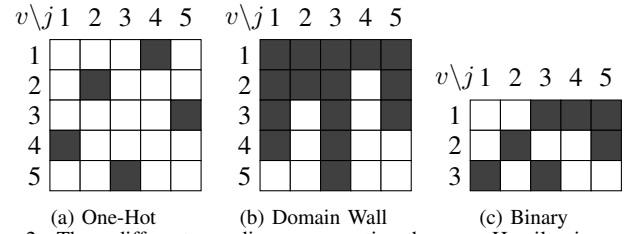


Fig. 2. Three different encodings representing the same Hamiltonian cycle on a fully connected graph consisting of five vertices.

Having that as a basis, a key concept for the construction of QUBO formulations is the definition of an encoding. While encodings can, in general, be defined specifically for particular problems, using general encodings has the advantage of a broader range of compatibility with individual problems. The specific encoding defines not only the number of binary variables required for the computation of the solution but also the complexity of the corresponding cost functions and the convergence behavior, depending on the employed algorithm. Therefore, supporting multiple different encoding types has the advantage of giving the approach more adaptability. Problems that cannot be solved efficiently using one encoding may, therefore, find solutions more easily in a different encoding. Through the proposed framework, the end user can easily test and reevaluate any of the supported encodings without the tedious reformulation process usually required for this task.

In the current version, the proposed framework supports three path-encoding types that each have individual advantages and disadvantages:

- *One-Hot Encoding* This encoding provides a set of $N \cdot |V| \cdot |\Pi|$ binary variables $x_{v,j,\pi^{(i)}}$. An individual variable with value 1 indicates that the corresponding vertex v is located at position j in path $\pi^{(i)}$. Assignments such that $x_{v,j,\pi^{(i)}} = x_{v',j,\pi^{(i)}} = 1$, with $v \neq v'$ are invalid.
- *Domain Wall Encoding* This encoding provides a set of $N \cdot |V| \cdot |\Pi|$ binary variables $x_{u,j,\pi^{(i)}}$. For a given vertex v to be located at position j in path $\pi^{(i)}$, the encoding requires $x_{u,j,\pi^{(i)}} = 1$ for all $u \leq v$. Compared to the one-hot encoding, which uses the same number of variables, this encoding has the advantage of allowing the transition between two valid states with just one bitflip, while the one-hot encoding requires two bitflips [34].
- *Binary Encoding* This encoding uses $\log N \cdot |V| \cdot |\Pi|$ binary variables. For a given vertex v to be located at position j in path $\pi^{(i)}$, the bitstring given by $x_{u,j,\pi^{(i)}}$, where $u \in V$, has to be the binary representation of v . The number of binary variables required for this encoding is asymptotically lower than that of the previous encodings, and there are no invalid assignments, at the cost of requiring more involved cost functions due to the complex nature of the encoding.

Example 2. Fig. 2 presents the example path $(v_4, v_2, v_5, v_1, v_3)$ in the three different encodings as tables, where the rows represent the index v and the columns represent the index j of $x_{v,j,\pi^{(i)}}$. Here, it is assumed that $|\Pi| = 1$ and, consequently, the index $\pi^{(i)}$ is omitted for the sake of presentation.

Once the end user selects the desired encoding through the user interface, the required constraints can be translated automatically into QUBO form before being combined into a single problem-specific cost function. For this matter, the framework provides support for a total of twelve constraints with pre-defined QUBO formulations for each encoding:

- 1) $\pi^{(i)}$ represents a valid path: Edges exist between any pair of consecutive vertices $v_{i_j}, v_{i_{j+1}}$ and the assignment follows the rules of the selected encoding.
- 2) The j -th element of $\pi^{(i)}$ is one of $\{v_1, v_2, \dots, v_k\}$.
- 3) The given *vertices* appear *at least once* in $\pi^{(i)}$.
- 4) The given *vertices* appear *exactly once* in $\pi^{(i)}$.
- 5) The given *vertices* appear *at most once* in $\pi^{(i)}$.
- 6) The given *edges* appear *at least once* in $\pi^{(i)}$.
- 7) The given *edges* appear *exactly once* in $\pi^{(i)}$.
- 8) The given *edges* appear *at most once* in $\pi^{(i)}$.
- 9) Given a set of paths, no two paths *share vertices*.
- 10) Given a set of paths, no two paths *share edges*.
- 11) Given two vertices u, v , enforce a *precedent constraint*: v may not appear in any path before visiting u .
- 12) The total weight of a set of paths is *minimall/maximal*.

In addition to that, the proposed framework also supports constraints that cannot be represented by quadratic cost functions (addressing another challenge discussed in Section III-A): depending on the complexity of the constraint and the selected encoding, some cost functions require higher-order products to be formulated. In these cases, the framework automatically extends the cost function by adding auxiliary variables. These auxiliary variables are created with the potential of reuse in mind. For each auxiliary variable, the framework stores the partial product it represents, and at later stages, higher-order products are first searched for partial products that have been encountered before and replaced by these variables. This greedy algorithm allows the number of required auxiliary variables to stay low.

As the task of predicting the number of required auxiliary variables for a particular problem instance on a given graph and with a pre-defined encoding is not trivial, the framework further supplies a utility to assist in the selection of encodings. For a given problem definition, it compares all supported encodings and evaluates which encoding requires the lowest number of binary variables for its QUBO formulation. This allows the user to easily select the most efficient encoding without requiring an understanding of any of the mathematical foundations.

C. Output Formats

After the required input has been provided to the framework and the automated QUBO formulation process is complete, one of several levels of output granularities can be selected for the representation of the resulting formula:

- As a single *Binary Cost Function*: A purely mathematical representation of the QUBO problem as a function $C(\mathbf{x})$ over the binary variables \mathbf{x} .
- As a *QUBO Matrix*: A triangular matrix Q that uniquely represents the QUBO problem as an optimization problem of the form $\arg \min_{\mathbf{x}} \mathbf{x}^T Q \mathbf{x}$.
- As a *Hamiltonian operator*: A quantum operator based on Qiskit that, when measured, yields the value of the corresponding QUBO cost function as its expectation value.
- As a *QAOA quantum circuit*: A Qiskit implementation of the QAOA algorithm, defined for the specific problem instance. Further QAOA parameters, such as repetition count, can be specified by the user.

This broad range of different output formats ensures a large variety of application areas. Not only can this framework be used to generate quantum computing solutions to pathfinding problems that are ready to be used on real quantum devices, but it also supports the generation of the QUBO formulation in its mathematical form so that it can be used with classical QUBO solvers, or as a Hamiltonian operator, so that it may

be coupled with different quantum algorithms. In all cases, the end user is provided with a QUBO formulation that has been generated automatically and did not require any QUBO or quantum computing background.

V. CASE STUDIES

In order to demonstrate the accessibility and utility of the proposed framework in solving various pathfinding problems, to confirm that the challenges discussed in Section III-A are addressed by it, and to evaluate the three different encoding schemes, several case studies on different problems have been conducted. In this section, the correspondingly obtained results are summarized. To this end, we first outline two of the considered case studies. Afterward, we discuss the results of these case studies, compare their performance among the different encoding schemes, and highlight key differences in the workflow with existing tools with similar goals.

The correspondingly used open-source implementations and GUI are available through <https://github.com/munich-quantum-toolkit/problemsolver>.

A. Case Study: Sequential Ordering Problem

In a first case study, we considered the *Sequential Ordering Problem (SOP)* which is defined by a graph $G = (V, E)$ and a set of *ordering constraints* $O = \{(v_{a_1} \prec v_{b_1}), \dots, (v_{a_k} \prec v_{b_k})\}$. It has the goal of finding the lowest-weight path π that visits all vertices in V exactly once while abiding by the ordering constraints: for each $(v_{a_i} \prec v_{b_i}) \in O$, vertex v_{b_i} may not be visited before visiting vertex v_{a_i} .

Given an SOP instance based on a fully connected graph consisting of five vertices and the ordering constraints $O = \{(v_1 \prec v_3), (v_2 \prec v_1), (v_5 \prec v_4)\}$, the problem is translated into its QUBO formulation using the proposed framework by breaking it down into the following constraints (denoted in brackets) from Section IV-B, i.e.,

- (4) all vertices $v \in V$ appear exactly once in π ,
- (11) v_1, v_2 , and v_5 appear before v_3, v_1 , and v_4 in π ,
- (12) $w(\pi)$ is minimal.

These constraints are passed to the proposed framework directly through its Python package or GUI. The GUI allows each of these constraints to be selected with the click of a button. It also allows for the easy selection of any desired encoding. After configuring the desired problem, the GUI generates a single JSON file which can be used as input for the Python package to generate a QUBO formulation as a cost function, a QUBO matrix or a quantum circuit. Note that this QUBO formulation is constructed without any need for decisions on binary variables or any mathematical expressions based on them, with the input based solely on the user's expertise on the specific pathfinding problem.

As an alternative approach, SOP problem instances can also be provided through the framework's *TSPLib* interface. This popular format for pathfinding problem specification allows the definition of various problem instances, including SOP. By passing a *TSPLib* specification of this problem to the proposed framework, its QUBO formulation can be constructed directly without any further user input.

Now, the QUBO formulation can be used as a basis for optimization algorithms to compute the optimal solution. By generating the output as an operator, it can be used directly by different quantum algorithms to perform the optimization.

B. Case Study: Two Disjoint Paths Problem

In a second case study, we considered the *Two Disjoint Paths Problem (2-DPP)* which is given by a graph $G = (V, E)$, two starting vertices v_{s_1}, v_{s_2} and two target vertices v_{t_1}, v_{t_2} . The

optimal solution to this problem is two paths, $\pi^{(1)}$ and $\pi^{(2)}$ with minimal total weight, starting and ending at v_{s_1}, v_{t_1} and v_{s_2}, v_{t_2} , respectively such that the paths do not intersect on any vertex (i.e., $V(\pi^{(1)}) \cap V(\pi^{(2)}) = \emptyset$ with $V(\pi)$ representing the vertices visited by path π).

This example considers, once again, a fully connected graph consisting of five vertices, with $(v_{s_1}, v_{t_1}) = (v_1, v_4)$ and $(v_{s_2}, v_{t_2}) = (v_2, v_5)$. Now, the QUBO formulation of this problem can be generated by the proposed framework by dividing it into the constraints, i.e.,

- (2) $\pi^{(1)}$ ($\pi^{(2)}$) starts at v_1 (v_2) and ends at v_4 (v_5),
- (9) $\pi^{(1)}$ and $\pi^{(2)}$ do not share any vertices, and
- (12) $w(\pi^{(1)}) + w(\pi^{(2)})$ is minimal.

Once again, each of these constraints can be specified directly in the GUI. While this problem instance requires constraints that are very different from the ones proposed for the SOP, it still does not involve any additional work to adapt the framework to the 2-DPP, as all required constraints are already supported by the framework.

Similarly to before, this formulation can then be used as the input for the quantum algorithms mentioned

C. Discussion

As shown by the two case studies, the burden on the end user for the construction of QUBO formulations has been drastically reduced in comparison to the solution using existing tools as reviewed in Section II-C and illustrated in Example 1:

- 1) No expertise in QUBO problems is required. The user-friendly GUI allows pathfinding problems to be described in the actual problem domain, while the end user is completely shielded from tasks involving QUBO or quantum computing concepts.
- 2) No mathematical expressions have to be created manually, avoiding a common source of errors and giving the user confidence in the generated QUBO formulation, even in complex cases.
- 3) The end user no longer needs to be concerned about the exact form of individual cost functions. Reformulations required for higher-order terms are performed automatically. This way, the user can select any supported constraint without the need to make sure it can be expressed using quadratic terms.
- 4) The end user is not required to manually merge all constraints into a single cost function, as this process is part of the framework's automated workflow.

This clearly confirms the accessibility and utility of the proposed framework and, indeed, addresses all the challenges discussed in Section III-A. While existing tools (as in Example 1) still require mathematical reformulation, our framework abstracts it entirely. This represents a deliberate trade-off: we sacrifice the open generality of mathematical programming to guarantee correct, simple, and automatic QUBO generation for the specific domain of pathfinding.

Moreover, the proposed framework provides value through its support of different encodings. This allows the user to easily evaluate individual QUBO formulations, switching between encodings at the click of a button, even as the resulting output changes disproportionately to the required effort. In comparison, even with the help of existing tools, a change of encoding requires users to redefine all constraints and their mathematic expressions, effectively requiring them to start the task from scratch for each new encoding.

In particular, the performance of individual encodings can be evaluated based on multiple factors:

TABLE II
REQUIRED VARIABLES FOR THE CASE STUDY PROBLEMS.

| Encoding | <i>One-Hot</i> | <i>Domain Wall</i> | <i>Binary</i> |
|----------|----------------|--------------------|---------------|
| SOP | 58 | 183 | 2375 |
| 2-DPP | 112 | 30 | 160 |

- *Variable Count*: Due to the nature of each encoding scheme, the number of variables required for each of them differs depending on individual scenarios. While the *Binary* encoding, for instance, requires the lowest number of variables to encode a given path, its complexity often requires higher-order terms to formulate cost functions and, thus, may lead to a large number of auxiliary variables. On the other hand, the *Domain Wall* and *One-Hot* encoding schemes often require just very simple cost functions, leading to fewer variables.
- *Convergence Behaviour*: As the encoding scheme directly correlates to the interpretation of a given variable assignment, optimization algorithms can pass through the optimization landscapes at different speeds depending on the chosen encoding. For instance, the *Binary* encoding does not lead to any invalid assignments. Any assignment can always be interpreted as a valid path. In the *One-Hot* encoding, on the other hand, the number of correct assignments is exponentially small compared to possible invalid assignments. This can impact both the convergence time as well as the likelihood of reaching the optimal solution.

Table II illustrates the variable requirements of both problems considered in the case studies above for all supported encoding schemes. This clearly shows the volatility of the encodings and the drastic improvements in variable count that can be achieved by selecting a different encoding. In the case of 2-DPP, the *Domain Wall* encoding did not require any auxiliary variables, as all higher-order terms were canceled out due to the nature of the encoding. Contrarily, in the case of SOP, the *One-Hot* encoding managed to require the lowest amount of variables, while the complexity of the *Binary* encoding resulted in more than 2000 auxiliary variables.

Thus far, evaluations like these have been a cumbersome and time-consuming task. While comparisons of different encoding schemes based on their performance exist for specific problem instances [34]–[38], the task of manually rewriting QUBO formulations for different encodings has proven to be too inefficient for thorough evaluation in most cases. In fact, once an end user struggled to get to one encoding and its corresponding QUBO formulation, they often did not bother to go through this hassle again for other encodings. Contrarily, using the proposed framework, different encodings can be evaluated at the click of a button.

VI. CONCLUSION

This work proposed an automatic framework to construct QUBO formulations from conceptually-defined pathfinding problems, shielding end users from the underlying mathematical and quantum-specific complexities. The framework allows users to select from a large number of constraints and, crucially, to easily switch between three supported encoding schemes. This circumvents the tedious and error-prone reformulation process, enabling users to investigate encoding performance and generate sound, qualitative results that can be passed directly to classical QUBO solvers or quantum algorithms such as QAOA or VQE.

The selection of constraints is supported through a Python package and a graphical user interface, both available open-source at <https://github.com/munich-quantum-toolkit/>

problemsolver. Additionally, established problem formats, such as *TSPLib* can be used to specify problem instances in familiar manners for domain experts. By automating the most difficult step of the optimization workflow, this work provides a practical tool to make quantum algorithms more accessible to domain experts and to aid quantum researchers in efficiently constructing and comparing correct problem formulations, addressing a significant challenge in the adoption of quantum computing.

Future work includes the addition of further constraints and encodings, a more detailed evaluation of performance metrics on real quantum computers for different encodings through *Resource Estimation* [39]–[45], and an integration of the framework into the growing hybrid quantum-classical ecosystem using intermediate representations such as MLIR and QIR [46], [47].

ACKNOWLEDGMENTS

This work received funding from the European Research Council (ERC) under the European Union’s Horizon 2020 research and innovation program (grant agreement No. 101001318), was part of the Munich Quantum Valley, which is supported by the Bavarian state government with funds from the Hightech Agenda Bayern Plus, and has been supported by the BMWK on the basis of a decision by the German Bundestag through project QuaST, as well as by the BMK, BMDW, the Austrian Research Promotion Agency (FFG) together with the states of Upper Austria and Tyrol within the COMET module Quantum Algorithm Engineering (FFG grant no. 923923), and the QuantumReady project within Quantum Austria (managed by the FFG).

REFERENCES

- [1] S. N. Kumar and R. Panneerselvam, “A Survey on the Vehicle Routing Problem and Its Variants,” *Intelligent Information Management*, 2012.
- [2] H. N. Psarafitis, M. Wen, and C. A. Kontovas, “Dynamic vehicle routing problems: Three decades and counting,” *Networks*, 2016.
- [3] T. P. Bagchi, J. N. D. Gupta, and C. Sriskandarajah, “A review of TSP based approaches for flowshop scheduling,” *European Journal of Operational Research*, 2006.
- [4] E. Alotaibi and B. Mukherjee, “A survey on routing algorithms for wireless Ad-Hoc and mesh networks,” *Computer Networks*, 2012.
- [5] P. E. Hart, N. J. Nilsson, and B. Raphael, “A formal basis for the heuristic determination of minimum cost paths,” *IEEE Transactions on Systems Science and Cybernetics*, 1968.
- [6] E. W. Dijkstra, “A note on two problems in connexion with graphs,” in *Edsger Wybe Dijkstra: His Life, Work, and Legacy*. Association for Computing Machinery, 2022.
- [7] T. Standley, “Finding optimal solutions to cooperative pathfinding problems,” *AAAI Conference on Artificial Intelligence*, 2010.
- [8] E. Erdem, D. Kisa, U. Oztok, and P. Schüller, “A general formal framework for pathfinding problems with multiple agents,” *AAAI Conference on Artificial Intelligence*, 2013.
- [9] J. K. Lenstra and A. H. G. R. Kan, “Some simple applications of the travelling salesman problem,” *Journal of the Operational Research Society*, 1975.
- [10] G. A. Kochenberger, F. Glover, B. Alidaee, and C. Rego, “A unified modeling and solution framework for combinatorial optimization problems,” *OR Spectrum*, 2004.
- [11] G. Kochenberger *et al.*, “The unconstrained binary quadratic programming problem: A survey,” *Journal of Combinatorial Optimization*, 2014.
- [12] N. Quetschlich, L. Burgholzer, and R. Wille, “Towards an automated framework for realizing quantum computing solutions,” in *Int’l Symp. on Multi-Valued Logic*, 2023.
- [13] D. Volpe, N. Quetschlich, M. Graziano, G. Turvani, and R. Wille, “Towards an automatic framework for solving optimization problems with quantum computers,” in *Int’l Conf. on Quantum Software*, 2024.
- [14] S. Kushnir, J. Leng, Y. Peng, L. Fan, and X. Wu, “QHDOPT: A software for nonlinear optimization with quantum hamiltonian descent,” *INFORMS Journal on Computing*, 2025.
- [15] H. Ushijima-Mwesigwa, C. F. A. Negre, and S. M. Mniszewski, “Graph partitioning using quantum annealing on the D-Wave system,” in *Workshop on Post Moores Era Supercomputing*, 2017.
- [16] L. Burgholzer *et al.*, “The Munich Quantum Software Stack: Connecting end users, integrating diverse quantum technologies, accelerating HPC,” in *Proceedings of the Supercomputing Asia and International Conference on High Performance Computing in Asia Pacific Region*, 2026.

- [17] A. Lucas, “Ising formulations of many NP problems,” *Frontiers in Physics*, 2014.
- [18] F. Glover, G. Kochenberger, R. Hennig, and Y. Du, “Quantum bridge analytics I: A tutorial on formulating and using QUBO models,” *Annals of Operations Research*, 2022.
- [19] J. D. Whitfield, M. Faccin, and J. D. Biamonte, “Ground-state spin logic,” *Europhysics Letters*, 2012.
- [20] M. Anthony, E. Boros, Y. Crama, and A. Gruber, “Quadratic reformulations of nonlinear binary optimization problems,” *Mathematical Programming*, 2017.
- [21] D. Rovara, L. Burgholzer, and R. Wille, “Quantum hardware-efficient selection of auxiliary variables for QUBO formulations,” in *Design, Automation and Test in Europe*, 2026.
- [22] E. Farhi, J. Goldstone, and S. Gutmann, *A quantum approximate optimization algorithm*, 2014. arXiv: 1411.4028.
- [23] A. Peruzzo *et al.*, “A variational eigenvalue solver on a photonic quantum processor,” *Nature Communications*, 2014.
- [24] T. Kadowaki and H. Nishimori, “Quantum annealing in the transverse ising model,” *Physical Review E*, 1998.
- [25] A. Gilliam, S. Woerner, and C. Gonciulea, *Grover adaptive search for constrained polynomial binary optimization*, 2019. arXiv: 1912.04088.
- [26] J. T. Iosue, “Qubovert.” (2020), [Online]. Available: <https://qubovert.readthedocs.io> (visited on 02/20/2026).
- [27] IBM, “IBM decision optimization CPLEX modeling for python.” (2023), [Online]. Available: <https://ibmdecisionoptimization.github.io/docplex-doc/> (visited on 02/20/2026).
- [28] Qiskit contributors, *Qiskit: An open-source framework for quantum computing*, 2023.
- [29] N. A. Wouda, L. Lan, and W. Kool, “PyVRP: A high-performance VRP solver package,” *INFORMS Journal on Computing*, 2024.
- [30] NVIDIA, “NVIDIA cuOpt.” (2025), [Online]. Available: <https://www.nvidia.com/en-us/ai-data-science/products/cuopt/> (visited on 02/20/2026).
- [31] D. Luxen and C. Vetter, “Real-time routing with openstreetmap data,” in *International Conference on Advances in Geographic Information Systems*, 2011.
- [32] R. Wille *et al.*, “The MQT Handbook: A Summary of Design Automation Tools and Software for Quantum Computing,” in *Int’l Conf. on Quantum Software*, 2024. arXiv: 2405.17543, A live version of this document is available at <https://mqt.readthedocs.io>.
- [33] G. Reinelt, “TSPLIB—a traveling salesman problem library,” *ORSA Journal on Computing*, 1991.
- [34] N. Chancellor, “Domain wall encoding of discrete variables for quantum annealing and QAOA,” *Quantum Science and Technology*, 2019.
- [35] J. Chen, T. Stollenwerk, and N. Chancellor, “Performance of domain-wall encoding for quantum annealing,” *IEEE Transactions on Quantum Engineering*, 2021.
- [36] P. Codognet, “Domain-wall / unary encoding in QUBO for permutation problems,” in *Int’l Conf. on Quantum Computing and Engineering*, 2022.
- [37] J. Berwald, N. Chancellor, and R. Dridi, “Understanding domain-wall encoding theoretically and experimentally,” *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 2022.
- [38] N. P. D. Sawaya, T. Menke, T. H. Kyaw, S. Johri, A. Aspuru-Guzik, and G. G. Guerreschi, “Resource-efficient digital quantum simulation of d-level systems for photonic, vibrational, and spin-s hamiltonians,” *npj Quantum Information*, 2020.
- [39] T. Hoeffler, T. Häner, and M. Troyer, “Disentangling hype from practicality: On realistically achieving quantum advantage,” *Communications of the ACM*, 2023.
- [40] M. E. Beverland *et al.*, *Assessing requirements to scale to practical quantum advantage*, 2022. arXiv: 2211.07629.
- [41] W. van Dam, M. Mykhailova, and M. Soeken, “Using Azure Quantum Resource Estimator for assessing performance of fault tolerant quantum computation,” in *Workshops of the International Conference on High Performance Computing, Network, Storage, and Analysis*, 2023.
- [42] M. Suchara, J. Kubiatowicz, A. Faruque, F. T. Chong, C.-Y. Lai, and G. Paz, “QuRE: The quantum resource estimator toolbox,” in *Int’l Conf. on Comp. Design*, Asheville, NC, USA, 2013.
- [43] N. Quetschlich, M. Soeken, P. Murali, and R. Wille, “Utilizing resource estimation for the development of quantum computing applications,” in *Int’l Conf. on Quantum Computing and Engineering*, 2024.
- [44] T. Forster, N. Quetschlich, M. Soeken, and R. Wille, “Improving hardware requirements for fault-tolerant quantum computing by optimizing error budget distributions,” in *Int’l Conf. on Quantum Computing and Engineering*, 2025.
- [45] T. Forster, N. Quetschlich, and R. Wille, “Quantum circuit optimization for the fault-tolerance era: Do we have to start from scratch?” In *Int’l Conf. on Quantum Computing and Engineering*, 2025.
- [46] C. Lattner *et al.*, “MLIR: Scaling compiler infrastructure for domain specific computation,” in *IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, 2021.
- [47] P. Hopf *et al.*, “Integrating quantum software tools with(in) MLIR,” in *Proceedings of the Supercomputing Asia and International Conference on High Performance Computing in Asia Pacific Region*, 2026.