Towards an Automated Framework for Realizing Quantum Computing Solutions

Nils Quetschlich* Lukas Burgholzer† Robert Wille*‡
*Chair for Design Automation, Technical University of Munich, Germany
†Institute for Integrated Circuits, Johannes Kepler University Linz, Austria
‡Software Competence Center Hagenberg GmbH (SCCH), Austria
nils.quetschlich@tum.de lukas.burgholzer@jku.at robert.wille@tum.de
https://www.cda.cit.tum.de/research/quantum

***Abstract*—Quantum computing is fast evolving as a technology due to recent advances in hardware, software, as well as the development of promising applications. To use this technology for solving specific problems, a suitable quantum algorithm has to be determined, the problem has to be encoded in a form suitable for the chosen algorithm, it has to be executed, and the result has to be decoded. To date, each of these tedious and error-prone steps is conducted in a mostly manual fashion. This creates a high entry barrier for using quantum computing—especially for users with little to no expertise in that domain. In this work, we envision a framework that aims to lower this entry barrier by allowing users to employ quantum computing solutions in an automatic fashion. To this end, interfaces as similar as possible to classical solvers are provided, while the quantum steps of the workflow are shielded from the user as much as possible by a fully automated backend. To demonstrate the feasibility and usability of such a framework, we provide proof-of-concept implementations for two different classes of problems which are publicly available on GitHub (https://github.com/cda-tum/MQTProblemSolver) as part of the Munich Quantum Toolkit (MQT). By this, this work provides the foundation for a low-threshold approach realizing quantum computing solutions with no or only moderate expertise in this technology.**

I. Introduction

Quantum computing devices are rapidly evolving and maturing with the increase of the number of available quantum computers as well as their number of qubits, error rates decreasing, and operations becoming faster. In parallel, numerous *Software Development Kits* (SDKs), such as Google's Cirq [1], IBM's Qiskit [2], Quantinmuum's TKET [3], and Rigetti's Forest [4], are being developed to make use of the available quantum computing hardware. Even specialized SDKs for certain purposes are available, e.g., Xanadu's Pennylane [5] for differentiable quantum computing. These developments spark interest in quantum computing from academia and industry—leading to potential applications in various domains such as physics [6], chemistry [7], finance [8], and optimization [9].

So far, many works aiming to solve specific problems by utilizing quantum computing follow a similar workflow consisting of four steps:

1) Selecting a suitable quantum algorithm.
2) Encoding the specific problem into a quantum circuit.
3) Executing it on a quantum device.
4) Decoding the solution from the quantum result.

While this has led to several promising quantum computing applications (triggering a substantial momentum for quantum computing in general), realizing the respective solutions comes with two major challenges: First, for all four steps, expertise in quantum computing is required. Without that, neither a quantum algorithm can be selected if the user is not aware of its prerequisites, nor can the problem be encoded, or the resulting quantum circuit be executed and the solution be extracted. Naturally, most of the users from those application domains are not trained experts in quantum computing which poses a huge roadblock in the further utilization and adoption of quantum computing. Second, especially during the encoding and decoding, many tedious and error-prone tasks have to be conducted—resulting in a huge manual effort to actually solve problems using quantum computing. Both aspects combined lead to a high entry barrier to employ quantum computing and make its utilization very challenging.

In this work, we envision a framework that simplifies the realization of quantum computing solutions—particularly for users from the various application domains. To this end, we exploit the fact that the current workflow summarized above actually offers tangible opportunities to shield the user as much as possible from the intricacies of quantum computing. This is accomplished by keeping the interfaces for both, the problem input and the solution output formats, as similar as possible to classical solvers and by providing guidance for the quantum algorithm selection procedure. Using this as a basis, the remaining steps (encoding, executing, and decoding) are then covered in a fully automated fashion.

To demonstrate the feasibility and usability of such a framework, a proof-of-concept implementation—which is publicly available on GitHub (https://github.com/cda-tum/MQTProblemSolver) as part of the *Munich Quantum Toolkit (MQT)*—has been realized for two different problem classes: *Satisfiability Problems* (SAT problems) and *Graph-based Optimization Problems*. For both, corresponding case studies confirmed the benefits from a user's perspective. By this, this work provides the foundation for a low-threshold approach of realizing quantum computing solutions with no or only moderate background in this technology.

The rest of this work is structured as follows: Section II gives a short introduction to quantum computing. Based on that, a detailed explanation of the mentioned workflow of realizing quantum computing solutions for specific problems is given in Section III. Afterwards, the tangible opportunities to automate and simplify this workflow are identified in Section IV—motivating the envisioned framework. Based on that, the proof-of-concept implementations are described in Section V and evaluated from a user's perspective in Section VI. Section VII concludes this work.

II. Quantum Computing

In order to keep this work self-contained, this section gives a short introduction to quantum computing. Compared to classical computing, where each bit can have a value of 0 or 1 representing its state, a quantum bit or *qubit* may also be in a *superposition* of those values, i.e., the *quantum state* $|\phi\rangle$ of a qubit can be written as

$$|\phi\rangle = \alpha_0 |0\rangle + \alpha_1 |1\rangle = \begin{bmatrix}\alpha_0 \, \alpha_1\end{bmatrix}^T$$

with amplitudes $\alpha_0, \alpha_1 \in \mathbb{C}$ such that $|\alpha_0|^2 + |\alpha_1|^2 = 1$. For n qubits, their state is composed of 2^n amplitudes $\alpha_i \in \mathbb{C}$ with i from 0 to $2^n - 1$. Again, the quantum state can be written as a superposition of all its basis states, i.e.,

$$|\phi\rangle = \sum_{i=0}^{2^n-1} \alpha_i |i\rangle = \begin{bmatrix}\alpha_0 \, \ldots \, \alpha_{2^n-1}\end{bmatrix}^T \text{with} \sum_i |\alpha_i|^2 = 1.$$

Analogously to classical computing and its logical gate operations, computations on quantum computers are conducted using quantum gates which alter the state of the qubit. The corresponding functionality of a quantum gate can be described by a *unitary* matrix; its effect on a quantum state can be determined by multiplying the matrix representation and the currently considered state representation.

Three prominent gates acting on a single qubit are the *Hadamard* (H), the *Pauli-X* (X), and the *Pauli-Z* (Z) gate which are defined by the matrices

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}, \; X = \begin{bmatrix}0 & 1\\ 1 & 0\end{bmatrix}, \text{ and } Z = \begin{bmatrix}1 & 0\\ 0 & -1\end{bmatrix}.$$

A prominent representative acting on two qubits (typically referred to as *control* and *target* qubits) is the controlled-not (CNOT) gate. If the control qubit is $|1\rangle$, the CNOT gate switches the amplitudes of the target qubit. In principle, any operation can be controlled by arbitrarily many qubits. A corresponding multi-controlled operation is only applied, if all control qubits are $|1\rangle$. An example for such a multi-controlled operation is the *multi-controlled-Z* (MCZ) gate. If all the control qubits are $|1\rangle$, the MCZ gate applies a Z gate to the target qubit. Those two operations with the second respectively the n^{th} qubit being the target qubit are defined by the matrices

$$CNOT = \begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 0 & 1\\ 0 & 0 & 1 & 0\end{bmatrix} \text{ and } MCZ = \begin{bmatrix}1 & 0 & \ldots & 0\\ 0 & \ddots & \ddots & \vdots\\ \vdots & \ddots & 1 & 0\\ 0 & \ldots & 0 & -1\end{bmatrix}.$$

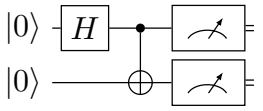


Fig. 1. Exemplary quantum circuit starting in state $|00\rangle$.

The state of a quantum system cannot be directly observed. Instead, a *measurement* collapses the state to one of the (classical) basis states $|i\rangle$—each with probability $|\alpha_i|^2$—which can then be read out.

Quantum algorithms are typically described in the form of a *quantum circuit*, i.e., a sequence of gates that are applied to the qubits of a quantum system.

Example 1. *Fig. 1 shows an exemplary quantum circuit with two qubits that first applies a Hadamard gate, followed by a CNOT gate. In the end, the qubits are measured.*

III. Realizing Quantum Computing Solutions

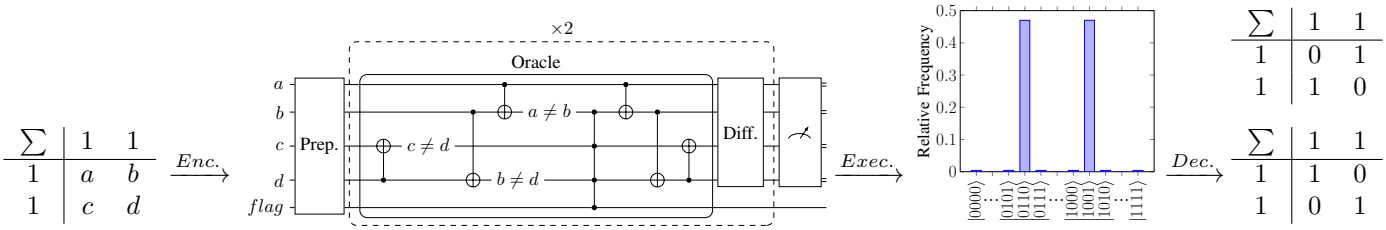
This section gives an overview of the main steps to be conducted when aiming to solve a specific problem using quantum computing. Based on that, the remainder of this work will then deal with how to automate this workflow or, at least, aid the user during this process. In order to properly illustrate those steps as well as the proposed solution, a running example is used, which is introduced in the following.

Example 2. *Kakuro, a cross-sum riddle, is an example of a SAT problem and is composed of a grid structure with M rows and N columns, where each row and column shall add up to a given sum. Additionally, numbers within each row and each column must be distinct. A simple Kakuro riddle instantiation with a grid structure of 2×2 is shown in Fig. 2a, where the goal is to determine a, b, c, and d such that the respective sums add up to 1. Thus, all those variables are to be assigned either 0 or 1. A solution to this problem is characterized by satisfying the constraints $a \neq b$, $b \neq d$, and $c \neq d$.*

A. Quantum Algorithm Selection

The first step towards solving a problem using quantum computing is to choose a proper quantum algorithm which is suitable for the considered problem. Without going into the details of quantum algorithms (for this, we refer to the given references), some prominent representatives are

- *Grover's Algorithm* [10], which is suited for unstructured search (problems),
- *Quantum Phase Estimation* (QPE, [11]), which provides the foundation of *Shor's Algorithm* [12] used for integer factorization, or
- *Variational Quantum Algorithms* (VQAs, [13]), e.g., *Variational Quantum Eigensolver* (VQE) and *Quantum Approximate Optimization Algorithm* (QAOA), which allows for hybrid classical-quantum computing while also being suited for combinatorial optimization problems, such as the *Max-Cut Problem*.



(a) Initial problem. (b) Quantum circuit. (c) Histogram. (d) Solution.

Fig. 2. Workflow from an initial problem instance to a valid solution using quantum computing.

Example 3. *Finding a solution for a SAT problem such as the Kakuro riddle can be interpreted as finding an assignment of all variables subject to some constraints. Being an unstructured search problem, Grover's algorithm is a very suitable choice for this. The corresponding quantum circuit is composed of three blocks: State preparation, a problem-specific oracle, and the diffusor. To utilize Grover's algorithm for solving a SAT problem, all constraints describing a valid variable assignment must be encoded within the oracle, while the state preparation and diffusor blocks are problem-independent. Together, the oracle and the diffusor constitute one* Grover iteration. *Each application of such an iteration increases the probability of obtaining correct solutions from the measurements—a technique called* amplitude amplification. *If n qubits are used and the underlying problem has k possible solutions, the maximal amplification is achieved after $\lfloor \frac{\pi}{4}\sqrt{\frac{2^n}{k}} \rfloor$ iterations.*

B. Encoding

After an algorithm is selected, the actual problem instance must be encoded in the form of a quantum circuit, such that the chosen quantum algorithm can determine a solution for that specific problem instance. The encoding fashion highly depends on the chosen algorithm and its inner working mechanisms. While some quantum algorithms, such as VQE and QAOA, follow a rather generic general fashion, others, such as Grover's algorithm, are far more problem-specific.

Example 4. *In order to apply Grover to the 2×2 Kakuro riddle shown in Fig. 2a, all three constraints mentioned in Example 2 need to be encoded in an oracle as described in Example 3. The resulting circuit is shown in Fig. 2b. Since each of the variables only assumes values in $\{0, 1\}$, a single qubit can be used to represent each of the variables a to d—amounting to four qubits. The three inequality constraints are then encoded through single CNOT gates (as annotated in Fig. 2b). Additionally, a* flag *qubit is introduced to indicate whether a variable assignment satisfies all constraints. An MCZ gate (indicated by the four black dots) is used for exactly this purpose and flips the phase of the flag qubit whenever all constraints are satisfied. Afterwards, all constraints are* uncomputed*—a prerequisite for Grover to function properly. With four qubits representing the open variables and two possible solutions, two Grover iterations are necessary (as indicated by the dashed rectangle and "$\times 2$" in Fig. 2b).*

C. Executing

Afterwards, the resulting quantum circuit is executed. In essence, executing a quantum circuit means repeatedly initializing a quantum system (typically to the all-zero state $|0...0\rangle$), applying all the operations of the circuit, and measuring the final state. This corresponds to sampling from the probability distribution described by the amplitudes of the final state and can either be performed on a classical computer (using *quantum circuit simulators* such as [2], [14]–[18]) or on actual quantum computers (such as from IBM, Rigetti, AQT, Google, Oxford Quantum Computing, or IonQ). In either case, the result after the execution is a histogram describing the distribution of the measured results.

Example 5. *The execution of the circuit from Example 4 yields an outcome distribution (referred to as a histogram) as illustrated in Fig. 2c. While most of the states have a very low probability of around 0.4%, the two highlighted states ($|0110\rangle$ and $|1001\rangle$) occurred in around 47% of the executions.*

D. Decoding

Based on the histogram obtained from the circuit execution, the solution to the problem must be decoded. In general, the solutions determined by the quantum algorithm are represented by the bitstrings in the histogram that occurred most frequently. The amount of post-processing necessary to transform these quantum solutions to classical solutions of the real problem greatly varies depending on the algorithm itself and the complexity of the encoding that was chosen to realize the quantum algorithm.

Example 6. *The two bitstrings 0110 and 1001 that occurred most frequently in the histogram shown in Fig. 2c encode the solutions to the 2×2 Kakuro riddle. Since every variable has been encoded as a single qubit, it is easy to read out the solutions $a = 0$, $b = 1$, $c = 1$, $d = 0$ and $a = 1$, $b = 0$, $c = 0$, $d = 1$. This eventually lead to the solutions shown in Fig. 2d.*

IV. Envisioned Framework

In this work, we envision a framework for developing quantum computing solutions that aims to *shield* users from the intricacies of quantum computing while at the same time providing them with all that is needed to realize their desired solution. To this end, the workflow as reviewed in Section III offers several tangible opportunities: To begin with, the original problem description is purely classical and, given

a suitable quantum algorithm, its encoding into a quantum circuit can be fully automated. In a similar fashion, also the execution and decoding step need not be handled by the user but can rather be embedded into an automated workflow as well. This yields the desired solution which can be returned to the user—again in a classical format. In doing so, all tedious and error-prone tasks involving huge manual effort are conducted by the framework itself instead of the user.

Overall, this leads to a framework as sketched in Fig. 3 which realizes a quantum computing solution for a given problem in four steps:

1) *Problem Specification:* The user is asked to classify the problem as well as to insert the particular instance of the problem. To this end, problems are grouped into overarching problem classes (such as SAT or graph-based optimization). For each problem class, corresponding interfaces are provided that allow the user to specify particular instances of the problem class. This is very similar to the inputs of classical solvers and, hence, does not require any quantum expertise at all.
2) *Algorithm Selection:* The user needs to select which quantum algorithm should be used to solve the problem. While this certainly requires some quantum computing expertise, the framework can actively support the user in this process. In fact, after choosing the problem class, often only a very selected number of algorithms remain appropriate to solve an instance of that class. For example, if a SAT problem has been specified, Grover's algorithm or QAOA seem to be the most promising algorithms. A corresponding a-priori selection can be conducted by quantum computing experts and, afterwards, accordingly incorporated into the framework. By this, the user is properly guided and shielded from the quantum domain as much as possible.
3) *Solving:* With the problem specification and the quantum algorithm set, the quantum computing solution can be realized. This requires the encoding, execute, and decoding steps reviewed above. Interestingly, all those steps can actually be conducted in an automatic fashion and, hence, completely shielded from the user. This builds on various demonstrations of problems being translated to the quantum realm, e.g., [6]–[9], and defers dealing with the intricacies of quantum computing from the user to the quantum computing expert developing the framework.
4) *Solution Processing:* Finally, the decoded solution is provided to the user in a format as it would have been provided by a classical solver.

This results in a *frontend* relying on classical descriptions and formats, where the user has to provide the problem instance and subsequently receives the solution; and a heavily automated *backend* that takes care of the encoding, executing, and decoding tasks. This keeps the user (who obviously only interacts with the frontend) in the classical domain (indicated by the green color in Fig. 3) and as much as possible shielded

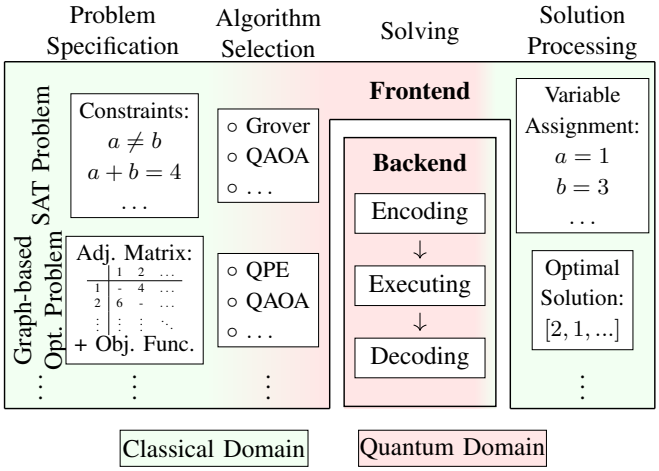


Fig. 3. Envisioned framework.

from the quantum domain (indicated by the red color in Fig. 3).

Eventually, this vision of a framework may enable users to utilize the benefits of quantum computing with no or only moderate background in this technology. However, implementing such a framework is no easy task: Interfaces, algorithm templates, automatic encoding, and decoding solutions, etc. need to be developed for a broad variety of problem classes. Moreover, naive implementations will certainly not be sufficient for practical use cases as they would lead to a substantial overhead today's quantum computers cannot handle. As a consequence, the development of *efficient* encodings and quantum circuit realizations [19]–[26] as well as methods properly compiling the resulting quantum circuits using the most promising compilation options [27] have received substantial interest recently. Overall, providing a fully-fledged implementation of the envisioned framework is clearly out of scope for this work and more of a long-term goal.

However, to allow for initial studies on the principal feasibility of the proposed vision, we realized two proof-of-concept implementations of the envisioned framework: One for the SAT problem class covered above; and one for the class of graph-based optimization problems. While these proof-of-concept implementations are certainly not optimized and scalable for practically relevant instances, they showcase that the concepts envisioned above are indeed feasible and can be extended to various problem classes. However, this comes with a large one-time development effort for each problem class for the quantum computing experts extending the framework.

V. Proof-of-Concept Implementations

In this section, we describe these proof-of-concept implementations. Based on that, Section VI then showcases that a framework as proposed above (even in simple proof-of-concept implementations) indeed allows for a realization of quantum computing solutions with no or only moderate quantum computing expertise.

A. Satisfiability Problems

As a first proof-of-concept implementation, we considered SAT problems and realized the respective components of the framework that are needed to provide support for this problem class. To this end, we created a Python skeleton defining the interfaces sketched in Fig. 3. Based on that, the respective components have been realized as follows:

First, we created the respective interface for SAT problems—requiring a dedicated input mask for defining the variables and constraints of arbitrary SAT instances. This interface is by no means different to the interfaces of classical tools from related domains and communities (e.g., based on SAT [28], [29], SMT [30]–[32], or ILP [33]). Hence, the huge set of description means, parsers, etc. developed in the past can be easily reused. In this implementation, we eventually decided to provide support for specifying small SAT problems formulated as a list of constraints.

Next, templates of possible quantum algorithms suited to tackle the problem are provided. As already discussed above in Example 3, Grover's algorithm is particularly suited for this class of problems while QAOA might also be a possible choice. For Grover's algorithm, a corresponding template is created, such that the general structure of the quantum algorithm is utilized for the subsequent encoding. This template consists of a state preparation, an oracle, and a diffuser (building) block.

To fill these blocks, automatic methods have been implemented that take the problem description in the form of the constraints list and construct the corresponding quantum circuit. More precisely, qubits are allocated for each variable in the constraints. From that, the state preparation and diffusion building blocks can already be fully instantiated (as they only depend on the number of qubits used). Then, the main task lies in encoding the given classical constraints in the form of an oracle, i.e., in terms of quantum gates. Since quantum gates (and quantum computing itself) are inherently reversible—while many classical functions are not—dedicated synthesis techniques need to be applied for that matter [34]. In our proof-of-concept implementation, each constraint is synthesized on a separate qubit employing existing techniques for reversible logic synthesis (such as quantum adders [35]–[37]). Then, all constraints are combined using an MCZ gate (similar to the encoding of the Kakuro riddle in Example 4).

After constructing the entire circuit from the given problem description in a fully automated fashion, it can now directly be passed to any device capable of its execution—either an actual quantum computer (e.g., from IBM, Rigetti, AQT, Google, Oxford Quantum Computing, or IonQ) or a classical quantum circuit simulator (e.g., [2], [14]–[18]). For the purpose of this work, we opted for the *MQT DDSIM*-simulator (taken from [15]) as a simple and easy to incorporate solution to generate the respective outcome distributions for the constructed circuits.

Since the number of solutions to the original problem might not be known a-piori (and, hence, the optimal number of Grover iterations cannot be statically determined), the proposed implementation dynamically adjusts the number of iterations until a definite result is obtained. Given the most probable measurement results, the respective variable assignments are then inferred in a similar fashion as in Example 6. Accordingly, the determined variable assignments are returned to the frontend, which visualizes them in a format familiar to the user.

B. Graph-based Optimization Problem: Travelling Salesman Problem

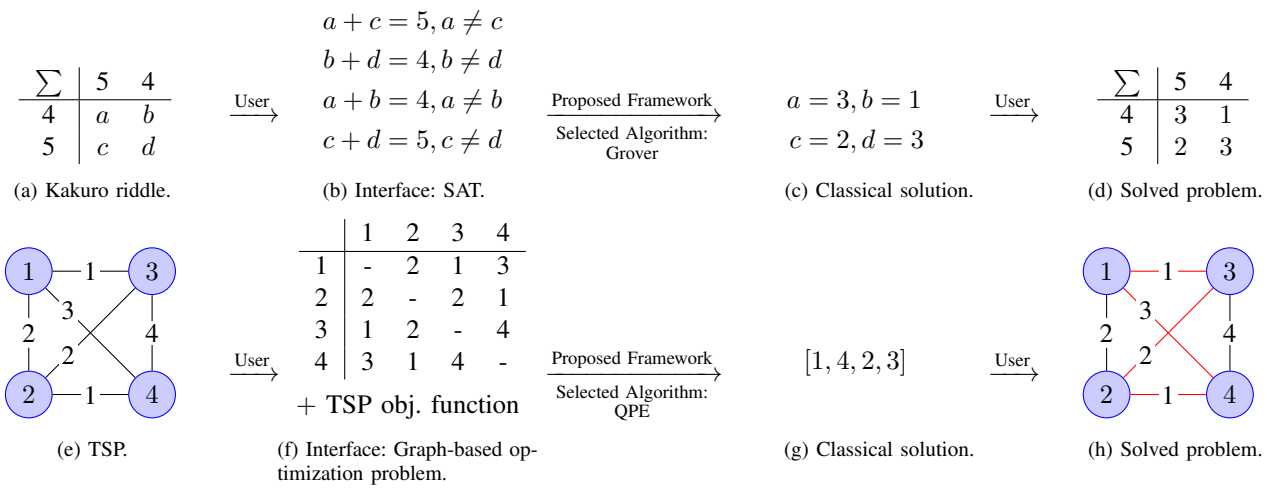
To demonstrate the flexibility and expandability of the envisioned framework, a proof of concept for a different problem class (namely graph-based optimization problems) has additionally been implemented.

Analogously to the previous implementation, we designed the respective interface for defining graph-based optimization problems by providing means to specify the graph-defining weighted adjacency matrix and the objective function. Again, this interface is similar to the interface of classical solvers for this type of problem.

For the proof-of-concept implementation, we particularly considered the *Travelling Salesman Problem* (TSP) as an objective function. In simple words, solving a TSP means to determine the shortest path visiting all nodes and ending at the starting node while each node is passed exactly once. More formally, the solution to the TSP is given by the shortest *Hamiltonian cycle* of the graph. The *Quantum Phase Estimation* (QPE, [11]) algorithm has been determined as a suitable technique for solving this problem on a quantum computer based on the encoding technique presented in [38].

The QPE algorithm allows to efficiently estimate the phase of one of a unitary matrix U's eigenvalues, i.e., estimates θ such that $U|\psi\rangle = e^{2\pi i\theta}|\psi\rangle$ for a given eigenstate $|\psi\rangle$. Similar to Grover's algorithm, where the encoding mainly concerns the design of the oracle part of the algorithm, the essential part of solving a problem with QPE is designing the unitary operator U so that the answer to the problem lies in the phase of its eigenvalues. For a TSP problem with N nodes, $N\lceil\log_2 N\rceil$ qubits are allocated to encode all possible Hamiltonian cycles. Next, the adjacency matrix is encoded by constructing a unitary matrix that encodes the respective weights as phases on its diagonal. By design, the Hamiltonian cycles of the graph (encoded as states $|\psi\rangle$) are eigenstates of this constructed operator U and the corresponding phase reflects the length of the cycle—the lower the phase, the shorter the cycle.

In the execution step, the QPE algorithm is applied to each unique Hamiltonian cycle—again utilizing the same quantum circuit simulator as before. Each execution run produces an estimate for the length of the corresponding Hamiltonian cycle. Accordingly, the path corresponding to the smallest measured phase is returned to the frontend as the solution to the problem—ready to be visualized and shown to the user.



(a) Kakuro riddle. (b) Interface: SAT. (c) Classical solution. (d) Solved problem.

(e) TSP. (f) Interface: Graph-based optimization problem. (g) Classical solution. (h) Solved problem.

Fig. 4. Case studies: Using the proposed framework to solve two problem instances of different problem classes.

VI. Case Studies: User's Perspective

Eventually, the steps described in the previous section led to a proof-of-concept implementation of the ideas proposed in Section IV which are publicly available on GitHub (https://github.com/cda-tum/MQTProblemSolver) as part of the *Munich Quantum Toolkit (MQT)*. In this section, the resulting framework realization is evaluated from a user's perspective in order to demonstrate that the proposed approach indeed allows one to utilize quantum computing for solving classical problems with no or only moderate quantum computing expertise. For that, problem instances from both the SAT and the graph-based optimization problem classes are exemplary solved.

A. SAT Problem: Kakuro Riddle

Consider again the Kakuro riddle that was used as a running example throughout this work. Fig. 4a shows a slightly more complex instantiation of such a riddle. The goal is to determine values for a to d such that the sums add up to 4 and respectively 5.

Using the proposed framework, the user first has to identify the corresponding problem class (here, obviously SAT) and provide the instance. For the latter, the classical constraints shown in Fig. 4b are passed to the framework and Grover's algorithm is selected as the quantum algorithm. All of that happens on a purely classical basis as it would have if the problem were to be solved using classical solvers. Afterwards, with the push of a button, the framework determines the variable assignment for a to d shown in Fig. 4c—completely shielding the user from all the tedious and error-prone quantum steps. The solved Kakuro instance is shown in Fig. 4d.

B. Graph-based Optimization Problem: TSP

In a similar fashion, the TSP shown in Fig. 4e can be automatically solved in a push-button fashion using the implemented framework. Remember, that the goal of the TSP is to determine the shortest path traversing all nodes exactly once and returning to the start.

All that is required from the user is the adjacency matrix of the graph and the selection of TSP as an objective function as shown in Fig. 4f—again sticking to purely classical and common description means. After selecting the QPE algorithm for solving the problem, with the push of a button, the framework determines the classical solution as reflected in Fig. 4g. This corresponds to the traversal of the graph as shown in Fig. 4h (indicated by the red edges).

VII. Conclusions

In this work, we envisioned a framework that allows users from various domains with little to no quantum computing expertise to use quantum computing for solving their problems. To achieve that, we shielded the user from most quantum computing steps and, instead, proposed corresponding automatic solutions for them. To this end, interfaces are employed that are as similar as possible to the interfaces of classical solvers for the respective problems. The feasibility and usability of such a framework has been demonstrated by proof-of-concept implementations and corresponding case studies for two different problem classes which are publicly available on GitHub (https://github.com/cda-tum/MQTProblemSolver).

While these case studies confirmed the potential of the envisioned framework, developing a fully-fledged implementation will require significant further efforts in many dimensions: Further problem classes need to be supported, generic implementations of further quantum algorithms need to be provided, and highly efficient encoding techniques need to be developed in order to allow the resulting framework to be of practical use. While this clearly is out of scope for this work and actually constitutes an ambitious long-term goal, this work lays the foundations for the design automation community to work towards this vision—eventually providing a low-threshold solution for utilizing quantum computing in real-world applications without the necessity of being an expert in quantum computing.

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, and the State of Upper Austria in the frame of the COMET program (managed by the FFG).

References

[1] C. Developers, *Cirq*, version v0.12.0, See full list of authors on Github: https://github.com/quantumlib/Cirq/graphs/contributors, 2021.

[2] M. S. ANIS *et al.*, *Qiskit: An open-source framework for quantum computing*, 2021.

[3] S. Sivarajah *et al.*, "T|ket⟩: A retargetable compiler for NISQ devices," *Quantum Science and Technology*, 2020.

[4] R. S. Smith, M. J. Curtis, and W. J. Zeng, *A practical quantum instruction set architecture*, 2016. arXiv: 1608.03355.

[5] V. Bergholm *et al.*, "PennyLane: Automatic differentiation of hybrid quantum-classical computations," 2018. arXiv: 1811.04968.

[6] A. Kandala *et al.*, "Hardware-efficient variational quantum eigensolver for small molecules and quantum magnets," *Nature*, 2017.

[7] A. Roggero *et al.*, "Quantum Computing for Neutrino-nucleus Scattering," *Physical Review D*, 2020.

[8] N. Stamatopoulos *et al.*, "Option Pricing using Quantum Computers," 2019. arXiv: 1905.02666.

[9] H. Mohammadbagherpoor *et al.*, "Exploring Airline Gate-Scheduling Optimization Using Quantum Computers," 2021. arXiv: 2111.09472.

[10] L. K. Grover, "A fast quantum mechanical algorithm for database search," in *Symp. on Theory of Computing*, New York, NY, USA: Association for Computing Machinery, 1996.

[11] A. Y. Kitaev, *Quantum measurements and the Abelian Stabilizer Problem*, 1995. arXiv: quant-ph/9511026.

[12] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM review*, 1999.

[13] M. Cerezo *et al.*, "Variational Quantum Algorithms," 2020. arXiv: 2012.09265.

[14] G. G. Guerreschi *et al.*, "Intel quantum simulator: A cloud-ready high-performance simulator of quantum circuits," *Quantum Science and Technology*, 2020.

[15] A. Zulehner and R. Wille, "Advanced simulation of quantum computations," *IEEE Trans. on CAD of Integrated Circuits and Systems*, 2019, MQT DDSIM is available at https://github.com/cda-tum/ddsim.

[16] S. Hillmich, I. L. Markov, and R. Wille, "Just like the real thing: Fast weak simulation of quantum computation," in *Design Automation Conf.*, 2020.

[17] T. Vincent *et al.*, *Jet: Fast quantum circuit simulations with parallel task-based tensor-network contraction*, 2021. arXiv: 2107.09793.

[18] B. Villalonga *et al.*, "A flexible high-performance simulator for verifying and benchmarking quantum circuits implemented on real hardware," *npj Quantum Information*, 2019.

[19] A. Zulehner and R. Wille, "Exploiting coding techniques for logic synthesis of reversible circuits," in *Asia and South Pacific Design Automation Conf.*, 2018.

[20] Y. Shee *et al.*, "Qubit-efficient encoding scheme for quantum simulations of electronic structure," *Physical Review Research*, 2022.

[21] B. Tan *et al.*, "Qubit-efficient encoding schemes for binary optimisation problems," *Quantum*, 2021.

[22] M. B. Dov *et al.*, *Approximate encoding of quantum states using shallow circuits*, 2022. arXiv: 2207.00028.

[23] A. Zulehner and R. Wille, "One-pass design of reversible circuits: Combining embedding and synthesis for reversible logic," *IEEE Trans. on CAD of Integrated Circuits and Systems*, 2018.

[24] B. Poggel *et al.*, *Recommending solution paths for solving optimization problems with quantum computing*, 2022. arXiv: 2212.11127.

[25] M. Soeken, R. Wille, and R. Drechsler, "Hierarchical synthesis of reversible circuits using positive and negative davio decomposition," in *International Design and Test Workshop*, 2010.

[26] S. Adarsh *et al.*, "SyReC Synthesizer: An MQT tool for synthesis of reversible circuits," *Software Impacts*, 2022.

[27] N. Quetschlich, L. Burgholzer, and R. Wille, *Predicting good quantum circuit compilation options*, 2022. arXiv: 2210.08027.

[28] G. S. Tseitin, "On the Complexity of Derivation in Propositional Calculus," in *Automation of Reasoning*, 1983.

[29] A. Biere *et al.*, *Handbook of Satisfiability*. IOS Press, 2009.

[30] L. de Moura and N. Bjørner, "Z3: An efficient SMT solver," in *Tools and Algorithms for the Construction and Analysis of Systems*, 2008.

[31] C. Barrett, P. Fontaine, and C. Tinelli, "The SMT-LIB standard: Version 2.6," Department of Computer Science, The University of Iowa, 2017.

[32] R. Wille *et al.*, "SWORD: A SAT like prover using word level information," in *VLSI-SoC: Advanced topics on systems on a chip*, 2009.

[33] G. L. Nemhauser and L. A. Wolsey, *Integer and Combinatorial Optimization*. Wiley, 1999, 763 pp.

[34] A. Zulehner and R. Wille, "Make it reversible: Efficient embedding of non-reversible functions," in *Design, Automation and Test in Europe*, 2017.

[35] V. Vedral, A. Barenco, and A. Ekert, "Quantum Networks for Elementary Arithmetic Operations," *Physical Review A*, 1996. arXiv: quant-ph/9511018.

[36] S. A. Cuccaro *et al.*, *A new quantum ripple-carry addition circuit*, 2004. arXiv: quant-ph/0410184.

[37] T. G. Draper, *Addition on a Quantum Computer*, 2000. arXiv: quant-ph/0008033.

[38] K. Srinivasan *et al.*, *Efficient quantum algorithm for solving travelling salesman problem: An IBM quantum experience*, 2018. arXiv: 1805.10928.