ADVANCED QUANTUM TECHNOLOGIES
www.advquantumtech.com

Qoolchain: A QUBO Preprocessing Toolchain for Enhancing Quantum Optimization

Giacomo Orlandi, Deborah Volpe, Mariagrazia Graziano, and Giovanna Turvani*

Solving combinatorial optimization problems is crucial in research and industry but still challenging since these problems are usually NP-hard or NP-complete. Classical solvers struggle with their non-polynomial complexity. Although heuristic algorithms are widely used, they often fall short in execution time and accuracy, increasing the interest in quantum computing alternatives using Quadratic Unconstrained Binary Optimization (QUBO) formulations. However, current Noisy Intermediate-Scale Quantum (NISQ) computers and future early fault-tolerant quantum devices face limitations in qubit availability and circuit depth, necessitating preprocessing to reduce problem complexity. This study introduces Qoolchain, a QUBO preprocessing toolchain designed to reduce problem size and enhance solver performance. Developed in Cython, Qoolchain is compatible with major quantum frameworks and optimized for the Grover Adaptive Search (GAS) algorithm. It includes steps like persistency identification, decomposition, and probing to estimate function bounds, all with polynomial complexity. Qoolchain also proposes using the Grover Search algorithm for problem segments whose optimal value is known a priori from graph theory and Shannon decomposition to reduce QUBO problem complexity further. Evaluated against the D-Wave preprocessing toolchain on various problems, Qoolchain demonstrates higher efficiency and accuracy. It represents a significant advancement in enabling practical quantum solvers, addressing hardware limitations, and solving complex industry-relevant problems.

1. Introduction

Optimization plays a strategic role in both research and industry contexts. It aims to identify the input configuration, minimizing or maximizing a merit expression, called objective function. If the inputs are discrete variables, it is defined as combinatorial. Applications of optimization span a spectrum of scenarios, enclosing tasks such as resource allocation[1–3] within an industrial setting and scheduling.[4]

Unfortunately, combinatorial optimization problems are classified as NP-hard or NP-complete, indicating that their classical solving approaches have a non-polynomial complexity. A naive and exact method for solving them is called brute-force and consists of evaluating the outcome with all possible input configurations. However, it is feasible from the computational point of view only for modest-size problems. Alternatively, a deterministic algorithm (e.g., based on gradient computation) can explore the solution space. Nevertheless, the approaches' effectiveness depends on the problem characteristics, e.g., they are unsuitable for multimodal objective functions, and reaching the convergence could be time-consuming. Heuristic algorithms, e.g., simulated annealing (SA),[5] are the most employed approaches for identifying optimal or sub-optimal solutions in large-scale problems.

Although numerous classical approaches have been introduced in recent years, they consistently fail to meet expectations regarding execution time or accuracy. Consequently, the interest in quantum alternatives has risen significantly, pushed by the captivating promise of obtaining a speed-up by exploiting its virtual intrinsic parallel exploration capability obtained through quantum principles such as superposition, entanglement, and tunneling.

For exploiting quantum-compliant approaches, the most feasible formulations are Quadratic Unconstrained Binary Optimization (QUBO) and Ising. Two paradigms have emerged in the quest for quantum advantage: Quantum Annealing (QA) and gate-based computing. The former leverages a special-purpose quantum computer theorized in 1998[6–10] and tries to exploit the inherent quantum system properties to find the optimal solution of an optimization problem. The latter is based on general-purpose quantum computers, where the state of quantum system evolution is stimulated by applying a sequence of unitary transformations called quantum gates.[11] This approach can be exploited with algorithms that are either fully or partially executed on these systems. Indeed, general-purpose quantum computers

G. Orlandi, D. Volpe, G. Turvani
Department of Electronics and Telecommunications of Politecnico di Torino
Torino 10129, Italy
E-mail: giacomo.orlandi@polito.it
M. Graziano
Department of Applied Science and Technology
Torino 10129, Italy

The ORCID identification number(s) for the author(s) of this article can be found under https://doi.org/10.1002/qute.202400384

DOI: 10.1002/qute.202400384

are currently employed as sub-routines to accelerate specific tasks of complex algorithms involving classical computers. Several algorithms have been proposed in recent years in this context. Among these, there is the Grover Adaptive Search (GAS),[12–15] which nicely illustrates the *accelerator* role played by the quantum computer. GAS progressively improves the solution by executing the Grover Search algorithm[16] for sampling increasingly negative values of a cost function. The cost function is shifted upward classically by the obtained sample at each iteration until the last negative is found. Other hybrid quantum-classical optimization algorithms are the Variational Quantum Eigensolver (VQE)[17] and the Quantum Approximate Optimization Algorithm (QAOA),[18] both based on variational routines. Contemporary Noisy Intermediate-Scale Quantum (NISQ) computers suffer from a severe limitation in terms of qubit (i.e. quantum bits) availability, and they are affected by non-ideal phenomena that become more relevant as the depth of the quantum circuit increases. Similarly, early Fault-Tolerant Quantum Computers (FTQC) expected in the future, for which GAS is designed to offer a significant speedup, will also need to address constraints in both qubit count and circuit depth. Therefore, for exploiting quantum approaches for relevant optimization problems, i.e., involving a high number of variables, preprocessing steps are needed to reduce the problem's complexity.

This article introduces **Qoolchain**, a QUBO preprocessing toolchain for reducing the problem complexity, making the exploitation of the quantum approaches feasible. The toolchain may be used for any optimization problem solver, but it has been thought to be employed, in its entirety, with GAS.

It is written in `Cython` language, i.e., with a `C++` core, ensuring high code efficiency. Its external `Python` interface makes it compatible with the main quantum framework. It comprises a set of steps with, at most, polynomial complexity, including persistency identification and problem decomposition for reducing the number of variables and the Probing technique for estimating the upper and lower bound of the problem cost function. Moreover, the decomposition could enable leveraging the Grover algorithm independently to address segments of the decomposed problem, whose minimum value is known in advance thanks to the theorem presented in ref. [19] and discussed in Section 4.3, resulting in faster attainment of that portion of the solution. Additionally, if prior steps are insufficient in reducing problem size for exploiting quantum solvers, Shannon decomposition is applied. Despite its exponential complexity, it has the potential to eliminate multiple variables with each subproblem, and its iteration-limited application significantly reduces its impact on the toolchain execution time. The results show that these strategies' effectiveness depends on the problem's peculiarity, such as the type of problem and the QUBO matrix density. The function's bounds obtained allow reasonable estimation of the necessary qubits for the GAS quantum circuit, the accuracy of which again depends on the problem's characteristics. Furthermore, this work proves the necessity of preprocessing steps for exploiting the current quantum solvers and, generally, finding the solution to more complex problems. It also highlights the main bottleneck (presented in Section 4.3) in the decomposition of QUBO problems.

The article is organized as follows. Section 2 presents the QUBO formulation, Grover Adaptive Search algorithm, and the related work in the literature. The proposed idea behind the toolchain and its structure are presented in Section 3, while it is analyzed step-by-step in Section 4. Section 5 shows the results, highlighting the relation between problem characteristics and the effectiveness of strategies. Finally, in Section 6, conclusions are drawn, and future perspectives are illustrated.

2. Theoretical Foundations

This section provides a concise overview of the foundational concepts in quantum computing and QUBO formulation, focusing on the GAS solver. Additionally, it presents an overview of the related work in the literature, emphasizing the gaps and unmet requirements within this context.

2.1. Quantum Computing Basis

Quantum computing represents a new computational paradigm, leveraging the principles of quantum mechanics, including superposition and entanglement, for accelerating data-intensive applications.

The qubit is the fundamental unit of information. Differently from the bit, its classical counterpart, which can either be in a state 0 or 1, the qubit can assume infinite possible states obtained as any linear combination of its basis states $|0\rangle$ and $|1\rangle$. Exploiting the Dirac notation, its state $|\psi\rangle$ can be described as a state vector as shown in the following:

$$|\psi\rangle = c_0\,|0\rangle + c_1\,|1\rangle = c_0\begin{pmatrix}1\\0\end{pmatrix} + c_1\begin{pmatrix}0\\1\end{pmatrix} = \begin{pmatrix}c_0\\c_1\end{pmatrix} \tag{1}$$

where $c_0, c_1 \in \mathbb{C}$ are, respectively, the amplitude probability associated with the states $|0\rangle$ and $|1\rangle$. The basis states form an orthogonal basis in the Hilbert space; thus, their linear combinations span the entire space. This numerical representation demonstrates that, owing to the superposition principle, a qubit can exist in any linear combination of its basis states, providing infinite possible states. Furthermore, the qubit state can be visually represented as a point on the surface of a unitary sphere called the **Bloch sphere**, shown in **Figure 1**, by expressing it in polar coordinates:

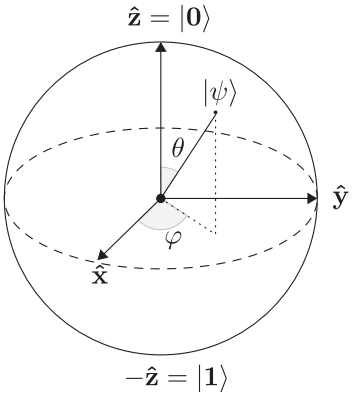
$$|\psi\rangle = \cos\frac{\theta}{2}\,|0\rangle + e^{i\phi}\sin\frac{\theta}{2}\,|1\rangle \tag{2}$$

where θ and ϕ are angles in the ranges $[0, 2\pi]$ and $[0, \pi]$, respectively.

However, for observing a qubit state, it is necessary to perform a measurement operation, inducing it to collapse into either of the two computational bases, $|0\rangle$ and $|1\rangle$, with a probability equal to the squared norm of each amplitude ($P(0) = |c_0|^2$, $P(1) = |c_1|^2$). The act of measurement forces the qubit into one of these two states, thus destroying the superposition. Since the two events are complementary, the following relation must be satisfied:

$$|c_0|^2 + |c_1|^2 = 1 \tag{3}$$

A n-qubit system can be described by expanding the numerical representation of a single qubit, writing its state vector $|\psi\rangle$ as the



$$\mathbf{r} = \begin{bmatrix} \sin\theta\cos\phi \\ \sin\theta\sin\phi \\ \cos\theta \end{bmatrix}$$

Figure 1. Representation of the qubit state on the Bloch sphere. On the right side, the Bloch vector shows the coordinates of the state vector $|\phi\rangle$ in the three-dimensional space.

tensor product of $|\psi_i\rangle$ of the single qubits:

$$|\psi\rangle = |\psi_{n-1}\rangle \otimes |\psi_{n-2}\rangle \otimes \cdots \otimes |\psi_1\rangle \otimes |\psi_0\rangle$$

$$= \begin{pmatrix} c_{0,n-1} \\ c_{1,n-1} \end{pmatrix} \otimes \begin{pmatrix} c_{0,n-2} \\ c_{1,n-2} \end{pmatrix} \otimes \cdots \otimes \begin{pmatrix} c_{0,1} \\ c_{1,1} \end{pmatrix} \otimes \begin{pmatrix} c_{0,0} \\ c_{1,0} \end{pmatrix} = \begin{pmatrix} c_{00\cdots00} \\ c_{00\cdots01} \\ \vdots \\ c_{11\cdots10} \\ c_{11\cdots11} \end{pmatrix}$$

$$= c_{00\cdots00}\,|00\cdots00\rangle + c_{00\cdots01}\,|00\cdots01\rangle + \cdots + c_{11\cdots10}\,|11\cdots10\rangle + c_{11\cdots11}\,|11\cdots11\rangle \tag{4}$$

where the probability amplitude $c_{00\cdots00}$ is associated with the basis state of the n-qubit system $|00\cdots00\rangle$, $c_{00\cdots01}$ to $|00\cdots01\rangle$ and so forth.

Quantum gates, formally described as a unitary matrix U (for which $UU^\dagger = U^\dagger U = I$) and performing reversible operations, must be applied to modify a qubit state. A sequence of gates forms a quantum circuit, as shown in **Figure 2**. The fundamental single qubit gates are the **Pauli gates** X, Y and Z, inducing a rotation of π along the x, y, z axis, respectively, and the **Hadamard gate** (H), which is exploited for creating the uniform superposition of the two bases states when applied to an input $|0\rangle$.

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \quad Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix},$$

$$Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}, \quad H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \tag{5}$$

Other fundamental gates are the rotation ones, allowing the rotation of the state vector around any direction by a parametric angle θ:

$$R_x(\theta) = \begin{bmatrix} \cos\frac{\theta}{2} & -i\sin\frac{\theta}{2} \\ -i\sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{bmatrix}, \quad R_y(\theta) = \begin{bmatrix} \cos\frac{\theta}{2} & -\sin\frac{\theta}{2} \\ \sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{bmatrix},$$

$$R_z(\theta) = \begin{bmatrix} e^{-i\frac{\theta}{2}} & 0 \\ 0 & e^{i\frac{\theta}{2}} \end{bmatrix} \tag{6}$$

Entanglement is a strong correlation between qubits, making the state of one dependent on the other. It can be created by applying gates involving at least two qubits. These gates are called **controlled gates**, and they do or do not perform an operation on a qubit or a set of qubits, called targets, according to the state of another qubit or set of qubits, referred to as controls. The most common gate in this category is the CNOT gate, a controlled version of the X gate that toggles the target only if the control is in the state $|1\rangle$. When the control qubit is the Least Significant Qubit (LSQ), and the target is the Most Significant Qubit (MSQ), its matrix is the following:

$$\text{CNOT} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \tag{7}$$

2.2. QUBO

Quadratic unconstrained binary optimization (QUBO)[20] is a mathematical formulation that allows the description of combinatorial optimization problems through a quadratic pseudo-boolean cost function. Every objective function involving binary variables can be reformulated into a polynomial form, reducible to second order, i.e., compliant with the QUBO formulation, at the expense of adding auxiliary variables.[21] Therefore, the QUBO model allows the representation of any combinatorial optimization problem and is inherently compatible with quantum solvers. The QUBO model can be described in two different ways. The first description can be obtained with the following objective function:

$$\text{minimize} \quad f(\mathbf{x}) = q_0 + \sum_i q_i x_i + \sum_{i<j} q_{ij} x_i x_j \tag{8}$$

where $x_i \in [0, 1]$ is a binary variable, $x_i x_j$ is a coupling term that allows two variables to influence each other, q_i is a weight or bias associated with a single variable, q_{ij} is a strength which controls the influence of variables i and j, whereas q_0 is an offset, which can be neglected during the optimization. Alternatively, the model can be expressed in matrix form:

$$y = \mathbf{x}^t Q \mathbf{x} \tag{9}$$

where $\mathbf{x}$ is the vector of binary variables, and Q is a square matrix (in upper triangular or symmetric form) in which the diagonal terms correspond to linear coefficients and the off-diagonal ones to quadratic.

Differently from what its acronym would suggest, the QUBO model can also take into account problems' constraints by integrating them into the initial cost function through weighted quadratic penalties:

$$\text{minimize} \quad f'(\mathbf{x}) = f(\mathbf{x}) + Pg(\mathbf{x}) \tag{10}$$

where $f(\mathbf{x})$ is the initial cost function, $g(\mathbf{x})$ the penalty function associated with the constraint, P a positive weight and $f'(\mathbf{x})$ the final objective function. A configuration of variables that does not breach any constraint results in a null penalty function, thus not

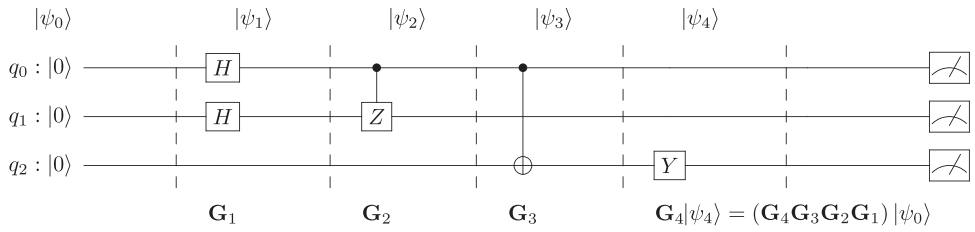


Figure 2. Example of a three-qubit quantum circuit. Vertical dotted lines highlight different layers. On the top, it shows how to compute the state vector evolution layer by layer, while the bottom right corner shows how the final state vector can be computed through the direct application of the overall circuit unitary matrix.

increasing the cost. The choice of P value is crucial since it should be high enough to penalize solutions that violate constraints but not so much to make it challenging to identify the optimal solution among the other valid ones. Indeed, a too-low value could lead to a non-valid solution, while an excessively high value could flatten the problem energy profile. Some techniques have been presented for estimating the optimal P based on the estimation of the initial cost function boundaries.[22–24]

For several commonly employed constraints, the penalty function is known a priori. Nevertheless, executing the following step makes it possible to obtain the penalty function for a generic constraint. Let us consider a generic optimization problem:

$$\begin{aligned} &\text{minimize} \quad f(\mathbf{x}) = \mathbf{x}^t C\mathbf{x}, \\ &\text{subject to:} \quad A\mathbf{x} = b \end{aligned} \tag{11}$$

The penalty function of a constraint expressed as a linear system can always be derived with the subsequent method:

$$f(\mathbf{x}) = \mathbf{x}^t C\mathbf{x} + P(A\mathbf{x} - b)^t(A\mathbf{x} - b) = \mathbf{x}^t C\mathbf{x} + \mathbf{x}^t D\mathbf{x} + c = \mathbf{x}^t Q\mathbf{x} + c \tag{12}$$

where the additive constant c can be neglected since it does not affect the solutions, but only their corresponding energy value (it behaves as an overall offset). All the constraints can be written as a linear system, eventually, as in the case of inequality constraint, at the expense of adding auxiliary variables (called **slack variables**), allowing the inclusion in the QUBO model. An attractive intrinsic feature of QUBO formulation is its equivalence to the **Ising model**,[25] a physical-mathematical model of ferromagnetism used in statistical mechanics. It allows the minimization of combinatorial optimization problems through systems inspired by the behavior of a physical system. The only difference between the two models lies in the nature of the binary variables involved—QUBO employs unipolar binary variables (0, 1), while the Ising model utilizes bipolar ones (−1, 1). Therefore, any problem formulated in QUBO can always be converted to the Ising model can to Ising by exploiting the following equivalence:

$$s_i = 2x_i - 1 \tag{13}$$

where s_i indicates an Ising variable, also called **spin**.

2.2.1. Benchmark Problems Considered

The problems considered for benchmarking the toolchain are presented in this section. They were chosen since they are sufficiently different from each other, allowing the evaluation of the approach's performance in different contexts and the identification of ones in which the toolchain application could be effective or not. QUBO formulation is written by employing the qubovert[26] library for each type of benchmark. In addition, for some of them, sets of problems available in the literature were exploited. This work considered a set of randomly generated problems of various sizes and densities for each benchmark. Specifically, for graph-based problems, the density is defined as follows:

Definition. *The density of a QUBO function (or matrix) is the fraction of the nonzero terms over all the possible quadratic terms. It measures how filled the QUBO matrix is with nonzero values and, consequently, offers an insight into the level of connectivity of the implication network based on the number of edges it comprises.*

Maximum Cliques: Given an undirected graph $G(V, E)$, with vertex set V and edge set E, a **clique** is the subgraph G', where an edge connects every vertex with all the other nodes in the subgraph. Therefore, the maximum clique's problem aims to identify the cliques involving the highest number of nodes, as shown in **Figure 3a**. The QUBO formulation can be written by associating a binary variable x_i for each vertex, which assumes the value 1 if it is part of the clique or is equal to 0 otherwise. Consequently, the problem can be written in the following way:

$$\begin{aligned} &\text{minimize } f(x) = -\textstyle\sum_{i\,in\,V} x_i, \\ &\text{subject to: } x_i + x_j \leq 1 \;\forall (i,j) \in \overline{E} \end{aligned} \tag{14}$$

where $\overline{E}$ indicates the set of couples of nodes not connected by an edge. The final cost function, including the constraint, is the following:

$$\text{minimize } f(x) = -\sum_{i \in V} x_i + P\left(\sum_{(i,j)\in\overline{E}} x_i x_j\right) \tag{15}$$

where the penalty coefficient is often chosen as $P = 2$. For estimating the toolchain performance with max clique, a set of randomly generated problems of different sizes and densities were considered.

Minimum Vertex Cover: The **Minimum Vertex Cover** problem[20] is an optimization job that involves an undirected graph with a set of vertices V and edges E. A **vertex cover** represents a **subset of vertices in which every edge is incident to at least one vertex from the subset**. Thus, the primary objective is to **identify the cover with the smallest number of vertices in the**

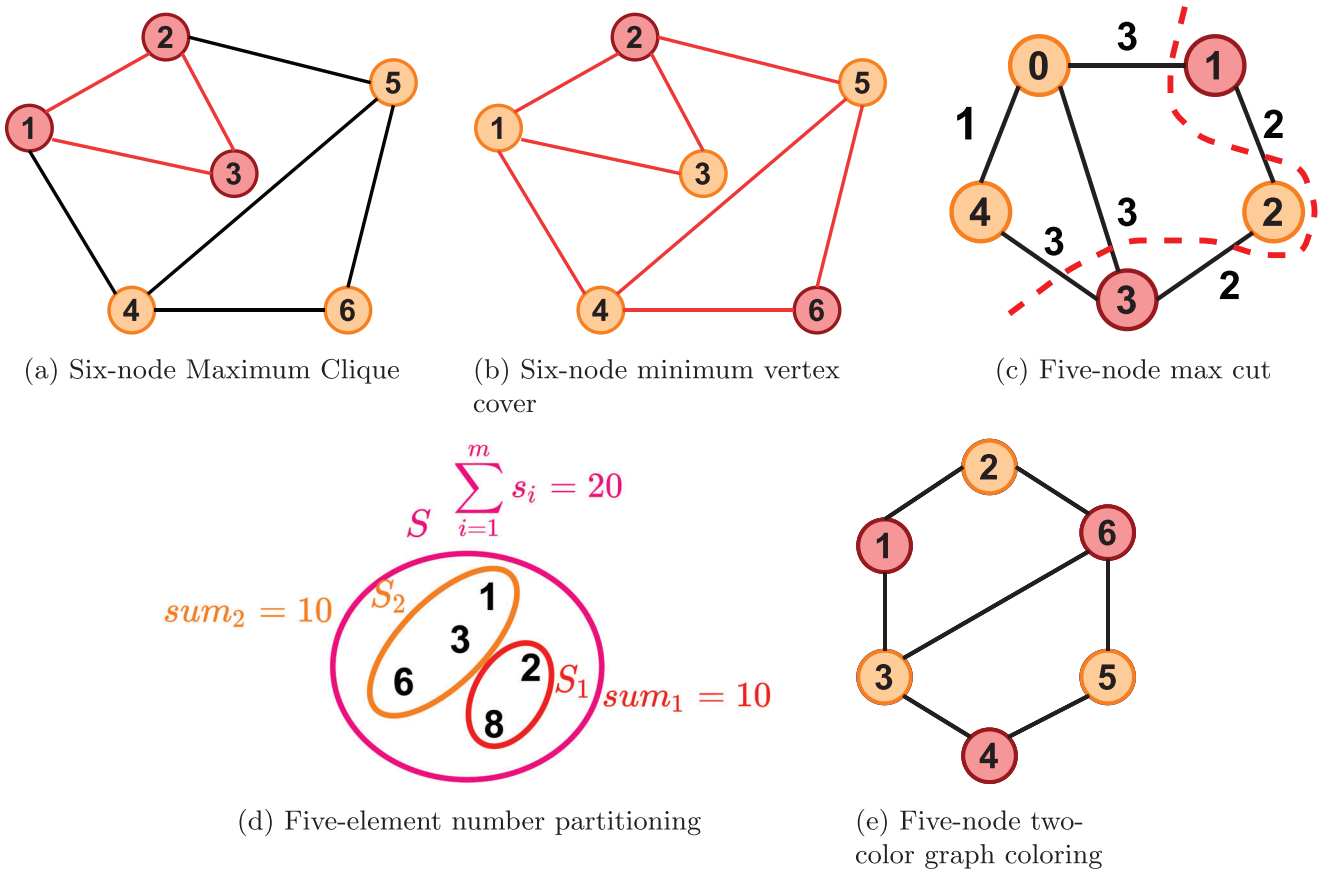


Figure 3. Optimization problems considered for benchmarking.

subset. For instance, in Figure 3b, the orange nodes constitute a vertex cover, as each edge in the depicted graph is connected to at least one of them, making it the optimal solution with the fewest involved nodes. With N vertices in consideration, the QUBO formulation introduces N **binary variables**, denoted as x_i, where each variable's value is one if the corresponding node is part of the cover. The final objective function consists of two components. The first component minimizes the number of nodes in the subset:

$$\text{Minimize} \quad y_1 = \sum_{i \in V} x_i \tag{16}$$

The second component ensures that each edge is incident to at least one vertex in the subset, enforcing the condition $x_i + x_j \geq 1$ for every pair of nodes i and j:

$$y_2 = \sum_{i,j \in E} \left(1 - x_i - x_j + x_i x_j\right) \tag{17}$$

Consequently, the complete objective function is expressed as:

$$f_{\text{vertex}}(\mathbf{x}) = \sum_{i \in V} x_i + P \sum_{i,j \in E} \left(1 - x_i - x_j + x_i x_j\right) \tag{18}$$

A set of randomly generated problems of different sizes and densities were considered to evaluate the toolchain performance with minimum vertex cover.

Max Cut: **Max cut**[27,28] (Figure 3c) stands out as one of the most noteworthy combinatorial optimization (CO) problems. It aims to **partition a graph into two complementary subsets, S and $\overline{S}$**, where **the sum of weights over all edges connecting the two vertex subsets is maximized**. This problem finds applications in several real-world scenarios such as network design, statistical physics, VLSI design, and circuit layout design.[29] The QUBO formulation of max cut involves **a binary variable for each node**, where the value is either 1 or 0, indicating the subset to which the node belongs. A **cut** is defined as **severing edges connecting nodes in different sets**. The quantity $\epsilon_{(i,j)} = x_i + x_j - 2x_i x_j$ determines whether the edge (i,j) is in the cut. Specifically, the edge (i,j) is part of the cut if $\epsilon_{(i,j)} = 1$, which occurs only when $x_i \neq x_j$. If both x_i and x_j are either 1 or 0, then $\epsilon_{(i,j)} = 0$. Considering the contributions of each edge, the objective function is expressed as follows:

$$\text{Maximize} \quad y = \sum_{(i,j) \in E} w_{i,j} \epsilon_{(i,j)} \quad = \sum_{(i,j) \in E} w_{i,j} \cdot (x_i + x_j - 2x_i x_j) \tag{19}$$

In this equation, $w_{i,j}$ denotes the weight of the edge connecting the i^{th} and j^{th} node. A remarkable characteristic of the max cut problem is its **symmetric energy profile**, i.e., solutions and their

complements (e.g., [0,1,1,0,1] and [1,0,0,1,0]) have the same energy, as the resulting subsets are interchangeable.

This work considers the well-known G-set max cut problems and a set of randomly generated problems of different sizes and densities for benchmarking purposes.

Number Partitioning: The objective of **number partitioning** optimization, as described in [20], is to divide a set $S = s_1, s_2, \ldots, s_m$ of m positive integers into two subsets S_1 and S_2 such that the sum of the numbers belonging each subset is equal. A graphical description is provided in Figure 3d. The QUBO formulation requires a binary variable x_i for **each number in the initial set** S, taking the value 0 if the i^{th} number is assigned to subset S_2 and 1 otherwise. As a result, the sum of the numbers in the first subset is expressed as:

$$\text{sum1} = \sum_{i=1}^{m} s_i x_i \tag{20}$$

while the sum of the numbers in the second subset is given by:

$$\text{sum2} = \sum_{i=1}^{m} s_i - \sum_{i=1}^{m} s_i x_i \tag{21}$$

Consequently, the difference between the two sums is:

$$\text{diff} = \text{sum2} - \text{sum1} = \sum_{i=1}^{m} s_i - 2\sum_{i=1}^{m} s_i x_i \tag{22}$$

which needs to be minimized to achieve the problem's objective. Thus, the final objective function is given by:

$$f_{\text{number}}(\mathbf{x}) = \left(\sum_{i=1}^{m} s_i - 2\sum_{i=1}^{m} s_i x_i\right)^2 \tag{23}$$

Similarly to the max cut problem, number partitioning exhibits **symmetric energy profiles**.

A set of randomly generated problems of different sizes and densities were considered for estimating the performance of the preprocessing techniques with the number partitioning task.

Graph Coloring: **Graph coloring**[20] is an optimization problem that aspires to **assign distinct color labels to neighboring nodes**, as represented in Figure 3e. This concept finds applications in various industrial and scientific domains, such as printed circuit design.[30] The associated QUBO formulation involves $N \cdot K$ **binary variables** — where N is the number of nodes and K the number of colors — , with one for **each node-color pair**, taking a value of one if the k^{th} color is assigned to the n^{th} node. Several requirements must be met to obtain a valid solution. Firstly, adjacent nodes must be associated with different colors:

$$\forall k: \quad \forall (i,j) \quad \text{adjacent node}: \quad x_{ik} + x_{jk} \leq 1 \tag{24}$$

expressible as:

$$\forall k: \quad \sum_{i,j \in E} x_{ik} x_{jk} = 0 \tag{25}$$

where E denotes the set of edges in the graph. Moreover, each node must be assigned precisely one color:

$$\forall n: \quad \sum_{k=1}^{K} x_{nk} = 1 \tag{26}$$

Finally, the objective function is defined as follows:

$$f_{\text{coloring}}(\mathbf{x}) = P_1 \sum_{n=1}^{N}\left(\sum_{k=1}^{K} x_{nk} - 1\right)^2 + P_2 \sum_{k=1}^{K}\sum_{i,j \in E} x_{ik} x_{jk} \tag{27}$$

A set of randomly generated problems of different sizes and densities was considered to estimate the toolchain effectiveness with graph coloring.

An alternative formulation of the graph coloring problem was proposed in the state-of-the-art to reduce the number of required variables, although this comes at the expense of introducing higher-order terms.[15] This approach is particularly advantageous when using solvers that can handle Polynomial Unconstrained Binary Optimization (PUBO) problems directly,[14] such as Grover Adaptive Search (GAS). However, since the proposed toolchain operates on QUBO problems and aims to be compatible with all quantum solvers—even if it offers additional support for GAS—the traditional formulation is exploited for benchmarking in this article.

2.3. Grover Adaptive Search

The Grover Adaptive Search is a sequential hybrid quantum-classical method for finding the minimum of a cost function $f(x)$, describing a combinatorial optimization problem.[12–15,31] It is composed of the following steps:

1. Initialize the iteration index i and the variable y_i to zero.
2. Adjust the cost function $f(x)$ by vertically shifting it based on y_i.
3. Increment the iteration index i.
4. Randomly select a value y_i from the range of $f(x)$ such that $y_i < 0$.
5. Treat y_i as the new optimal value, as a negative value of the adjusted cost function consistently indicates a lower value than the previously set offset y_{i-1}.
6. Repeat steps 2–5 until no more negative values are sampled. This process identifies the combination of variables where $f(x) = 0$, which is the solution to the optimization problem.

A significant example of this approach is reported in **Figure 4**, where it is possible to notice how the minimum of the cost function can be identified by repeating the sample-and-shift procedure.

An effective negative sampling method must be identified to exploit the presented approach adequately. The best option for this charge is the **Grover Search (GS)**[32] routine, a quantum approach for searching items in an unordered dataset with a quadratic speed-up compared with the classical counterpart. The flowchart of the approach is reported in **Figure 5**, where the task separation between the classical and quantum worlds is underlined.

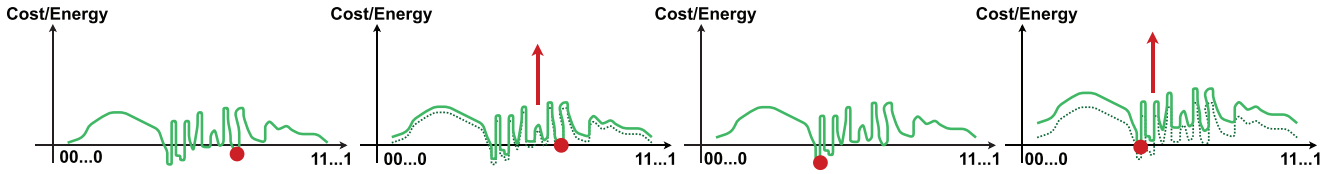


Figure 4. GAS steps.

Initially, the objective function shifted by an amount y_i has to be encoded into a quantum state as a uniform superposition of the domain-image obtained through the employment of a quantum dictionary,[14] which is the quantum counterpart of a classical dictionary and allows the description of key-value pair data structures. It is important to remember that the $f(\mathbf{x})$ image is described according to the binary two's complement representation for signed integers. In general, cost function coefficients can also be expressed as real values by exploiting the technique described in ref. [33]. Therefore, for encoding an n-variable problem, n qubits for the keys, and m — sufficiently high to avoid overflow, so depending on the function bounds—for the values are required. Unfortunately, the computational complexity for exact estimation of the cost function bounds is the same as solving the problem with a brute-force approach, i.e., grows as 2^n. Proper estimation techniques are required since a too-low value compromises the mechanism's effectiveness, and a too-high is a waste of resources. Preprocessing is thereby necessary for choosing a proper value of m.

The following equation describes the quantum dictionary operator U_i:

$$U_i |0\rangle_n |0\rangle_m = \frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} |x\rangle_n |f(x) - y_i\rangle_m \tag{28}$$

At this point, the Grover phase oracle is applied for labeling the desired items of the dataset—in this case, the negative samples of the objective function—and the diffuser operator is employed for amplifying the probability amplitude of the corresponding states. For implementing an effective GS, phase oracle and diffuser have to be repeated r times, where r is also called "number of rotations," and its optimal value depends on the dataset characteristics. It can be estimated through the following equation:

$$r \approx \frac{\pi}{4}\sqrt{\frac{2^n}{M}} \tag{29}$$

where M is the number of desired items, i.e., the negative samples, which is unknown a priori. This number of rotations ensures finding one of the M states with the maximum probability[11]; hence, either a too-high or too-low r may lead to an incorrect result. Its estimation in each GS execution is crucial, and some techniques were proposed in the literature for this purpose.[12] The quantum circuit employed for GS execution in the GAS algorithm is reported in **Figure 6**.

Once a negative value has been acquired, the function can be classically shifted by that amount and re-encoded in the quantum dictionary for searching a new negative. The samples obtained are, with each iteration, closer to the function's minimum until the last negative was obtained, corresponding to the optimal solution. However, determining whether negative values are still to be sampled is a crucial task. Theoretically, a characteristic of GS can be used for this goal: when no item meets the conditions of the GS, any possible configuration can be sampled according to a uniform probability distribution. If only non-negative function values are available, the GS outcomes are positive or null samples. Unfortunately, there are other situations where a non-negative value can be obtained since it can also emerge when an incorrect number of Grover rotations is chosen. Consequently, proper techniques, principally based on the

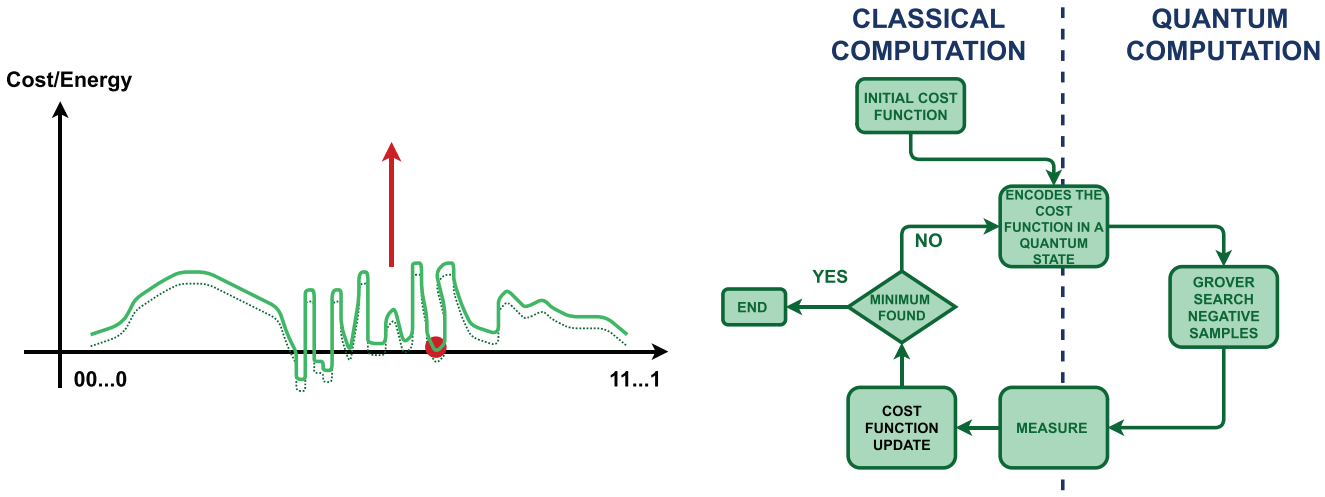


Figure 5. GAS algorithm.

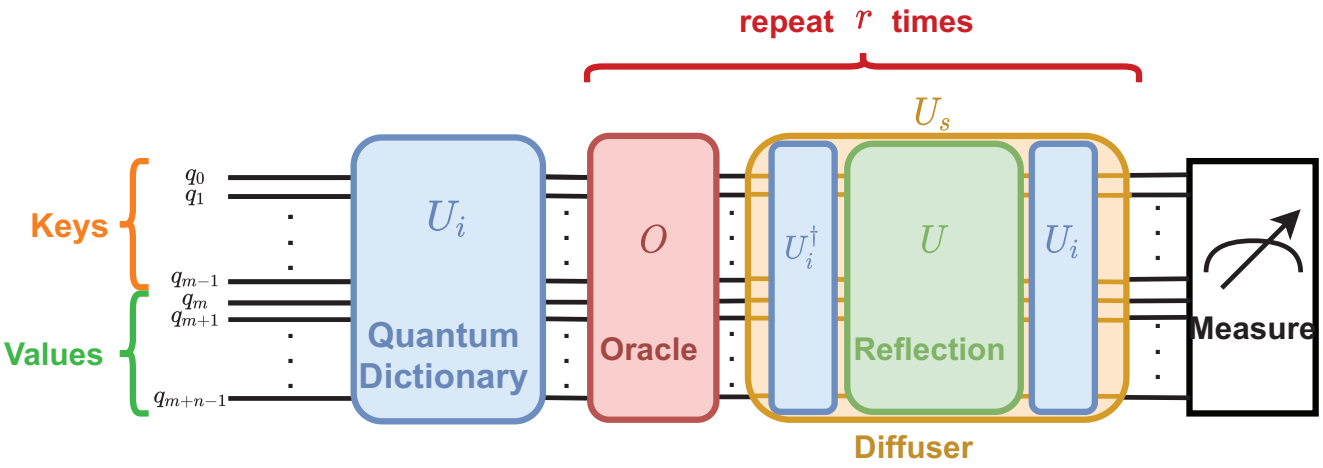


Figure 6. GAS circuit.

counts of consequently positive samples and the employment of thresholds, were defined in the state of the art for stopping the algorithm.[12]

2.4. QUBO Preprocessing: State of the Art

In recent years, D-wave has released the first QUBO preprocessing toolchain[34] optimized for their Quantum Annealer platform, including methods for fixing persistencies and estimating the lower bound of the objective function based on approaches presented in ref. [35]. This represents the only previous work in this direction. In addition to D-Wave preprocessing methods, Qoolchain provides the following tools:

1) the Probing technique presented in ref. [35] for identifying further persistence and improving the estimation of the function bounds;
2) the Trivial decomposition, proposed in this article for the first time, for subdividing the problem into independent subproblems of smaller size;
3) decomposition enabling the employment of Grover Search for identifying the optimal solution of a sub-graph whose optimal value is known a priori through graph theory properties;
4) the Shannon decomposition, proposed in this article for the first time, for breaking Complete Strongly Connected Components (defined in 4.1.4), allowing further reduction of the QUBO problem.

Table 1 provides a comprehensive comparison between D-wave preprocessing and Qoolchain.

3. Towards the Idea of the QUBO Preprocessing Toolchain

This section presents the motivations behind a QUBO preprocessing toolchain targeting the Grover Adaptive Search algorithm. Moreover, it comprehensively overviews the key preprocessing steps and delves into the framework's structure.

3.1. Motivations

Assessing the potential of quantum solvers, especially those based on the quantum circuit model paradigm like GAS, in identifying the optimal solution to real-world optimization problems poses significant challenges. Indeed, nowadays, quantum hardware is strongly limited in terms of qubit counts, connectivity, and fidelity. Therefore, the prevalent approach for developing quantum solutions involves leveraging quantum simulators, which offer ideal outcomes unaffected by current hardware limitations. Unfortunately, the simulation complexity grows exponentially with the number of involved qubits, making the execution of large circuits impractical. All these aspects inhibit the exploration of quantum computing-based solutions for real-world problems.

The toolchain introduced in this article aims to address these challenges by providing instruments to streamline the complexity of QUBO models, thereby making the execution of quantum solvers, especially GAS, feasible even for optimization problems encountered in industrial environments. Specifically, the toolchain endeavors to achieve this goal through:

1) the reduction of the number of QUBO variables of the problem to solve, fixing those whose optimal solution can be a

Table 1. Comparison of D-wave preprocessing and Qoolchain. * identifies the method proposed in this article for the first time; ✓ indicates the supported approaches; ✗ signifies that the method is not supported.

Tools	Source-reachable persistencies	Strongly connected persistencies	Probing	Trivial Decomposition*	Grover Search solving*	Shannon Decomposition*
D-wave	✓	✓	✗	✗	✗	✗
Qoolchain	✓	✓	✓	✓	✓	✓

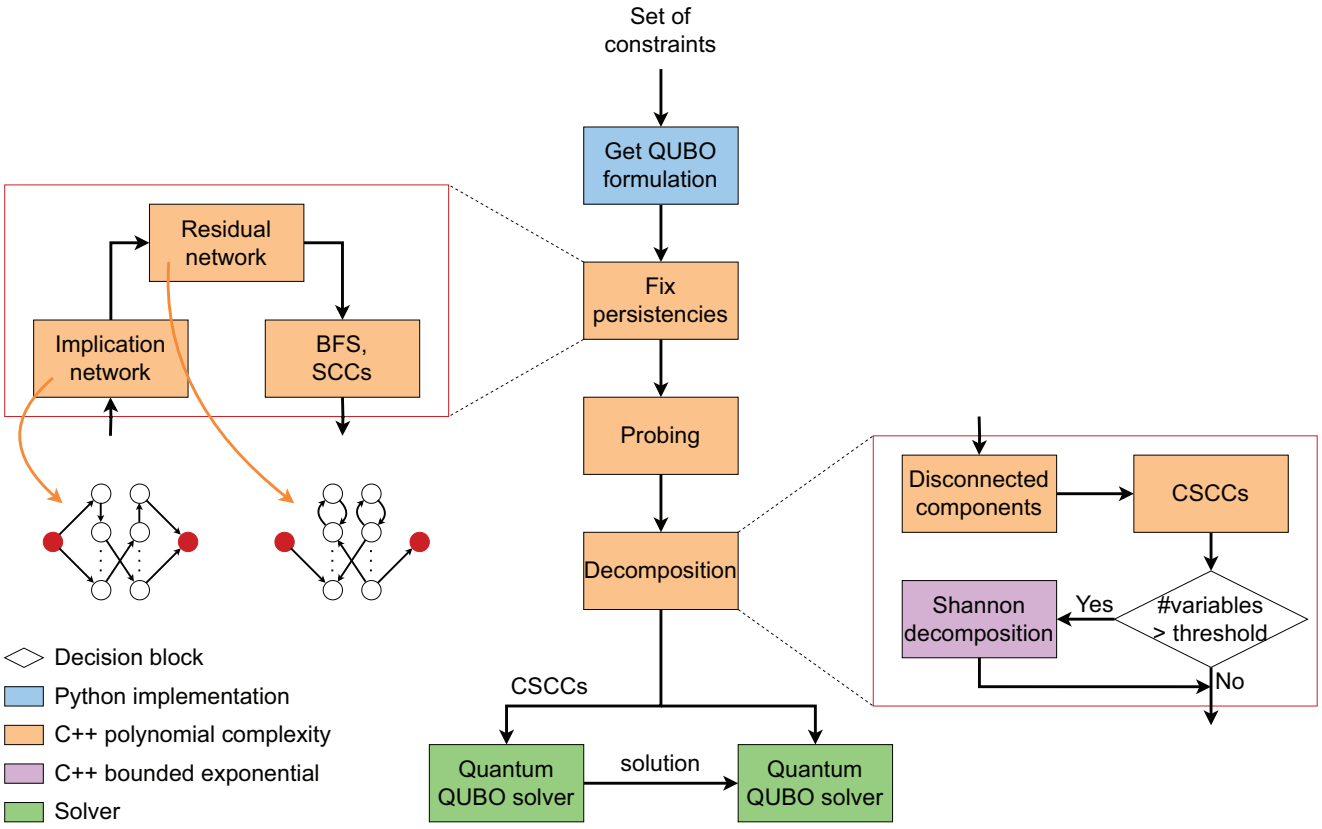


Figure 7. Toolchain structure overview.

priori identified (persistency) and decomposing problems in smaller subproblems;
2) the estimation of the function bounds for choosing the number of qubits for values representation in proper GAS.

While the toolchain is designed to enhance accessibility to quantum solutions within the current hardware limitations, it is expected to retain its utility even as quantum computers surpass existing constraints. Indeed, effective QUBO preprocessing, in general, overcomes the limits of the target solver, allowing the evaluation of even bigger problems.

3.2. Structure

The toolchain comprises several stages, analyzing and reducing the QUBO formulation of interest, as shown in **Figure 7**. The toolchain requires the optimization problem expressed according to the QUBO formalism as input, exploiting the qubovert[26] library or providing the QUBO matrix in its upper triangular form.

First of all, the input QUBO problem is analyzed for identifying the persistencies—i.e., all the binary variables whose optimal configuration can be detected a priori[35]—through the *Fix persistencies* block. Afterward, their optimal configuration can be substituted in the cost function, reducing its size. As detailed in Section 4.1, the recognition of those variables, through the application of a proper algorithm, requires the transformation of the QUBO function into a graph representation called flow network, consisting of a directed graph whose edges have non-negative weights where it is possible to distinguish a source node with only outgoing edges and a sink with only ingoing edges. Subsequently, the *Probing* technique, detailed in Section 4.2, estimates the cost function bounds and, consequently, the number of qubits necessary for encoding the cost function in a quantum dictionary. The application of this technique potentially allows the identification of further persistencies.

Afterwards, a *Decomposition* step, presented in Section 4.3, is applied. It comprises three main methods. The first finds disconnected components in the networks, which correspond with independent subfunctions. The second identifies and extracts Complete Strongly Connected Components (CSCCs), which are a particular kind of strongly connected components (SCCs) that can be encountered in residual networks, whose formal definition is given in Section 4.1.4. Finally, the Shannon decomposition technique is only employed when the problem size exceeds a predefined threshold. In analogy with the Shannon expansion for Boolean functions, it selects a variable and generates two subnetworks, each with one less variable. However, all previous methods can be applied again to the new subnetworks aiming to break CSCCs for which no method is currently known.

Finally, the residual networks are transformed back to QUBO function representations to be provided as inputs for the last step composed of the Grover Adaptive Search, or in general, any optimization solver. CSCC and CSCC-free functions can be solved

Table 2. Implemented method computational complexity. n is the number of graph nodes; m is the number of graph edges; t is a user-defined threshold, i.e., the number of repetitions of the Shannon Decomposition procedure.

Residual Network	Source-reachable persistencies	Strongly connected persistencies	Probing	Trivial Decomposition	Shannon Decomposition
$\mathcal{O}(n^2\sqrt{m})$	$\mathcal{O}(n+m)$	$\mathcal{O}(n(n+m))^{*}$	$\mathcal{O}(n^3\sqrt{m})$	$\mathcal{O}(n)$	$\mathcal{O}(2^t)^{**}$

* For strongly connected persistencies for each SCC, a BFS is executed, hence just in the worst case the number of SCCs is $n/4$, but it is not likely to happen. ** The execution of the Shannon decomposition is limited by a predetermined threshold t which has been set to 5 in all the tests reported in this paper.

in two different instances, considering that once the former are solved, the minima of the latter are already known, as discussed in Section 4.3 (**Table 2**).

4. Implementation

The core of the proposed toolchain has been developed in `C++`, leveraging object-oriented programming principles. As the main quantum frameworks are mostly written in `Python`, the external interface for calling its functions has been implemented in `Python`, facilitating user-friendly interactions with other quantum tools. Specifically, `Cython`[36] has been exploited to establish a bridge between `C++` and `Python`. `Cython` enables the compilation of `C` or `C++` functions into `Python` extension modules, thereby enabling the development of tools that combine the speed of compiled languages with the user-friendly and flexible interface of `Python`.

`Cython`, derived from `Pyrex`, is a language designed for creating `Python` extension modules that abstract the `Python/C` API from the user. This abstraction allows users to easily combine `Python` and `C` data structures without requiring knowledge of the `Python/C` API. To encapsulate `C++` functions within a `Pyrex` file (or a `Python` file), it is essential to follow the workflow shown in **Figure 8**. This process involves creating a wrapper file — which translates function declarations from the header file into `Cython` syntax, converts parameters from `Python` data structures to `C++`, and returns `C++` objects to `Python` ones — and compiling the `C++` sources with a proper setup script. The outcome is a standard `Python` module that can be called within any `Python` script.

This section presents the implemented toolchain, focusing on each step composing it, and providing relevant examples.

4.1. Persistencies

In ref. [35] persistencies have been defined as follows:

Definition. *Let $a \in \mathbb{B}$ be a binary value, a variable x_i is a **strong persistency** if $x_i = a$ for all the minima of the objective function, whereas it is a **weak persistency** if $x_i = a$ for at least one minimum of the function.*

In order to identify persistencies, the QUBO function has to be transformed into an **Implication network**, i.e., a particular type of flow network, introduced in ref. [37]. The peculiarity of a flow network is that it is possible to define a function that assigns a value to each edge no larger than its weight, known as **flow** (φ).[38] A flow function must fulfill specific properties:

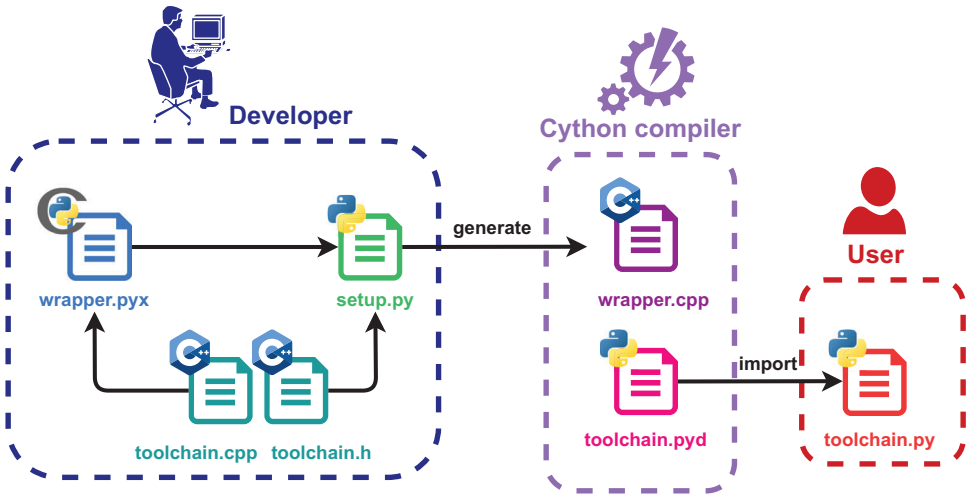


Figure 8. Graphical representation of the procedure for obtaining a Python module with a `C/C++` core by exploiting the `Cython` language. First, the developer writes the `C/C++` function of interest (`.cpp` and `.h`) and the `Pyrex` file with the header file containing the declaration of the function of interest (`.h`). In the `Pyrex` file, the target function has to be translated into `Cython` syntax, and the `C++` function must be called within a wrapper function that converts its inputs from `Python` data structures to `C++` types and converts the returned from `C++` types to `Python` data structures. Then, the `setup.py` file, which specifies the name of the `Python` extension to obtain and the `C++` and `Pyrex` files to consider, is exploited for compiling the `C++` source with `Cython`. All the files produced by the developer are shown in the magenta box. The compilation produces an optimized translation of the wrapper (`_wrapper.cpp`), and the `Python` module to import in other scripts (`.pyd`). These files are shown in the green box. The obtained module can be imported by the user as a standard `Python` module, as illustrated in the orange box.

- The flow from a node u to a node v is equivalent to the negation of the flow from v to u ($\varphi(u,v) = -\varphi(v,u)$).
- The total amount of flow entering a node, except source and sink, must be the same as the amount of flow exiting the node.

Subsequently, the flow allows transforming a flow network into a **residual network**. This is the data structure on which all the toolchain algorithms operate. Translating the QUBO cost function into a residual network requires several steps that are explored in the following sections.

4.1.1. Implication Network

The first stage is purely mathematical and allows obtaining a quadratic expression with only nonnegative coefficients, except eventually a constant term, called **posiform**. Starting from a QUBO cost function, the result is a quadratic posiform. However, the same method can be applied to any polynomial expression. In general, any polynomial pseudo boolean function can be transformed into a posiform with linear complexity scaling with the number of nonzero coefficients. All the steps for translating a QUBO function into a posiform are shown in ref. [35]. The obtained pseudo-boolean function has the following form:

$$\phi(x) = c_o + \sum_{u\in L} c_u u + \sum_{u,v\in L} c_{uv} uv \tag{30}$$

where L is the set of literals, i.e. the set of all the variables and their complements, and $c_u \geq 0$ and $c_{uv} \geq 0$, for any $u, v \in L$. Note that it is possible from a posiform to go back to a quadratic pseudo-boolean function using the inverse procedure.

Example. *Let's consider the following QUBO function:*

$$f(x) = 3 + 2x_1 + 3x_2 - x_3 + x_4 + x_1x_2 - 4x_2x_3 + x_3x_5 - 2x_4x_5 \tag{31}$$

First of all, to convert the function in a posiform, the $q_{ij}x_ix_j = q_{ij}x_i(1 - \overline{x_j}) = q_{ij}x_i - q_{ij}x_i\overline{x_j}$ *transformation, where is* $\overline{x_j}$ *is the* x_j *variable complement, has to be applied to every negative quadratic coefficient:*

$$\phi(x) = 3 + 2x_1 + 3x_2 - x_3 + x_4 + x_1x_2 - 4x_2(1 - \overline{x_3}) + x_3x_5 - 2x_4(1 - \overline{x_5}) \tag{32}$$

$$= 3 + 2x_1 - x_2 - x_3 - x_4 + x_1x_2 + 4x_2\overline{x_3} + x_3x_5 + 2x_4\overline{x_5} \tag{33}$$

Now, for every linear term with a negative coefficient, the substitution $x_i = 1 - \overline{x_i}$ *is applied:*

$$\phi(x) = 3 + 2x_1 - (1 - \overline{x_2}) - (1 - \overline{x_3}) - (1 - \overline{x_4}) + x_1x_2 + 4x_2\overline{x_3} + x_3x_5 + 2x_4\overline{x_5} \tag{34}$$

$$= +2x_1 + \overline{x_2} + \overline{x_3} + \overline{x_4} + x_1x_2 + 4x_2\overline{x_3} + x_3x_5 + 2x_4\overline{x_5} \tag{35}$$

The obtained expression is the posiform of the QUBO problem of interest.

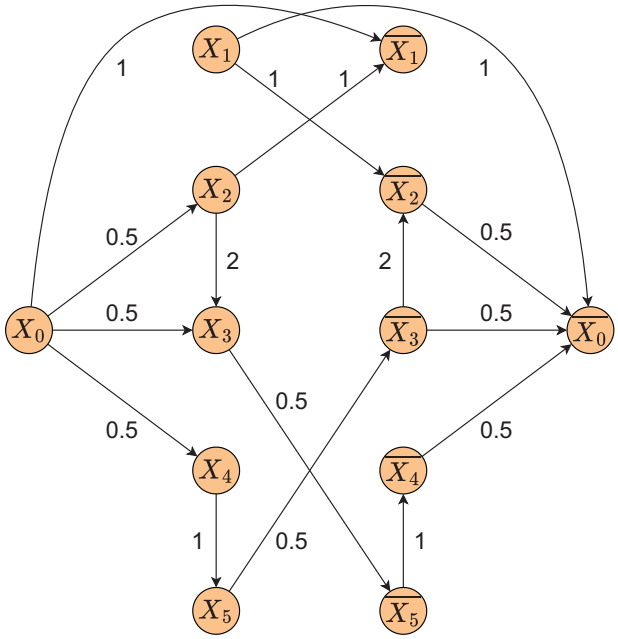


Figure 9. Example of implication network associated with the QUBO cost function $f(x) = 3 + 2x_1 + 3x_2 - x_3 + x_4 + x_1x_2 - 4x_2x_3 + x_3x_5 - 2x_4x_5$.

A posiform can be expressed as an implication network. However, it has to be transformed into a purely quadratic function, i.e., including only quadratic terms. For this purpose, all the linear terms are multiplied by a fictitious variable x_0 with a fixed assignment $x_0 = 1$. Instead, The constant term can be moved to the left-hand side of the equation and just be considered an offset. For each term $c_{uv}uv$, there are two edges, $(u, \overline{v})$ and $(v, \overline{u})$, in the implication network, having a weight equal to half of their coupling coefficient (**Figure 9**).

$$\gamma(u, \overline{v}) = \gamma(v, \overline{u}) = \frac{1}{2}c_{uv} \tag{36}$$

Since the literal x_0 has only outgoing edges and the literal $\overline{x_0}$ has only ingoing edges, they can be called the network source and sink, respectively. The name of this graph representation comes from the fact that if a literal u receives the assignment $u = 1$, it implies that all literals reached through an edge starting from u must also be assigned to 1. This prevents an increase in the posiform's value; otherwise, uv would equal 1. Conversely, if u receives the assignment $u = 0$, all literals capable of reaching node u must be assigned to 0; otherwise, the term $v\overline{u}$, corresponding with the edge (v, u), would equal 1.

Example. *Considering the previous example:*

$$\phi(x) = +2x_0x_1 + x_0\overline{x_2} + x_0\overline{x_3} + x_0\overline{x_4} + x_1x_2 + 4x_2\overline{x_3} + x_3x_5 + 2x_4\overline{x_5} \tag{37}$$

The network associated with this purely quadratic function is shown in Figure 9. For instance, the term $2x_0x_1$ *produces two edges* $(x_0, \overline{x_1})$ *and* $(x_1, \overline{x_0})$ *both with weight* $c_{x_0x_1}/2 = 1$.

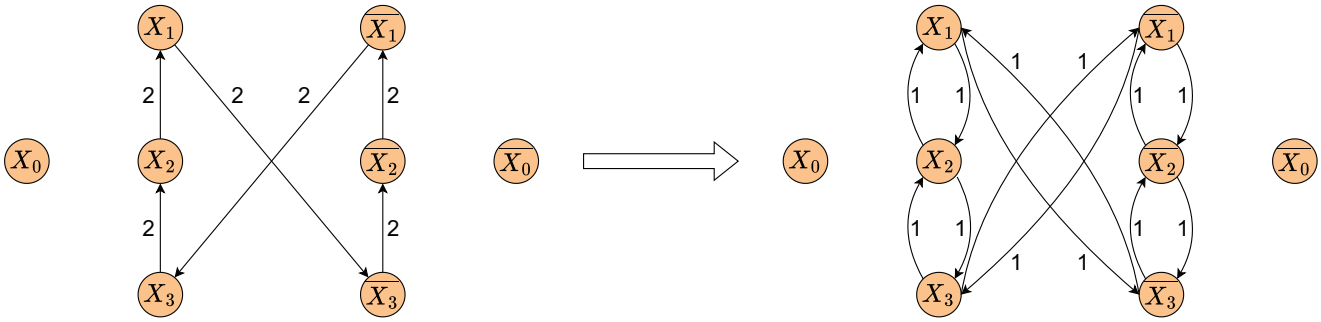


Figure 10. Example of residual network symmetrization.

4.1.2. Residual Network

Definition. *Given a flow network with node set N and edge set E and indicating the flow of an edge (u, v) as $\varphi(u, v)$, a residual network is a network with the same node set N and edge set E_φ with residual weight $\gamma_\varphi(u, v) = \gamma(u, v) - \varphi(u, v)$. Since a property of the flow is that $\varphi(u, v) = -\varphi(v, u)$, if an edge (u, v) was present in the implication network, an edge (v, u) can be created in the residual one.*

The maximum flow is the flow for which there are no paths from source to sink in the relative residual network. In order to attain this residual network, the maximum flow is computed through the **push-relabel method**[39,40] with the **highest-label** implementation which has $\mathcal{O}(n^2\sqrt{m})$ computational complexity,[41] where n is the number of nodes and m is the number of edges in the network. The obtained residual network may be asymmetrical, i.e. $\gamma_\varphi(u, v)$ may be different from $\gamma_\varphi(\overline{v}, \overline{u})$. However, the implication network has a one-to-one correspondence with its posiform. This relation should be kept in the residual network since it has to be eventually transformed back into a QUBO function. Therefore, for each edge (u, v), the final weight is given by the average between its weight and the weight of its symmetrical counterpart.

$$\gamma_\varphi(u, v) = \gamma_\varphi(\overline{v}, \overline{u}) = \frac{1}{2}(\gamma_\varphi(u, v) + \gamma_\varphi(\overline{v}, \overline{u})) \tag{38}$$

In all the residual network representations throughout this paper, edges entering the source and exiting the sink are not depicted since they have no role in all the algorithms presented. Furthermore, in this article, referring to the residual network always indicates the residual network constructed with the maximum flow.

Example. *The previous example forms an already symmetrical residual network. However, considering the following cost function:*

$$f(x) = 4 + 4x_2 - 4x_1x_2 + 4x_1x_3 - 4x_2x_3 \tag{39}$$

*The residual network obtained after constructing the implication network and computing the flow is reported on the left in **Figure 10**. As it is possible to notice, the symmetrization step is required in this case. For instance, $\gamma_\varphi(x_1, x_2) = 0$, but $\gamma_\varphi(\overline{x_2}, \overline{x_1}) = 2$ and the same can be observed for any edge in this graph. The symmetrization result is shown on the right in Figure 10*

4.1.3. Source-Reachable Persistencies

Each variable connected directly or indirectly to the source can be assigned to 1. For this reason, this type of persistencies will be called **source-reachable persistencies** from herefore. To prove that each source-reachable node is a persistency, proof by contradiction can be provided. Let us consider a subset S collecting the nodes that can be reached with a path from the source x_0, where all the edges have positive weights. Defining $T = \{\overline{v} | v \in S\}$, the following reflections can be derived:

- If a quadratic term $a_{uv}uv$ in the posiform has $a_{uv} > 0$, the nodes u and v cannot both belong to S since it implies the existence of an edge $(u, \overline{v})$ with positive weight. If $v \in S$, then $\overline{v} \in T$, thus implying the existence of a path from source to sink, violating the maximality of the flow. This condition is impossible in a residual network by definition. Therefore, if a node is connected to the source, it is always possible to find an assignment eliminating the terms in which it appears.
- Linear terms $a_u\overline{u}$ produces an edge (x_0, u) in the network, which can be eliminated by assigning $u = 1$. Consequently, all the literals in S are strong persistencies since the assignment that sets them to 1 makes all the terms in which they are involved vanish. This is also in agreement with the definition of implication network for which all the nodes reachable from a node assigned to 1 must also be assigned to 1.

These **source-reachable** persistencies can be identified by using a **breadth-first search** (**BFS**) starting from the source to mark all the visited nodes.

4.1.4. Strongly Connected Persistencies

Definition. *The **strongly connected components** (**SCCs**) K_i (also called strong components) of a graph are subgraphs in which every node can be reached by any other node. Due to the symmetry of residual networks, if a node v belongs to a strong component, it can be deduced that either $\overline{v} \in K_i$ or $\overline{v} \in K_i'$, where K_i' is the component containing the complements of all the literals belonging to K_i.*

Definition. ***Complete strongly connected components** (**CSCCs**) are strong components including both the literal v and its complement $\overline{v}$.*

All the variables included in a strong component not belonging to the CSCC category are weak persistencies. In particular,

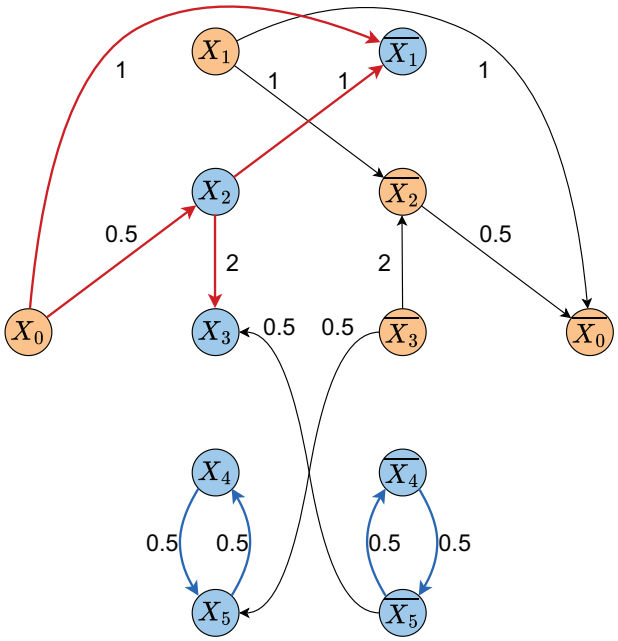


Figure 11. Example of source-reachable and strongly connected weak persistencies. The former are highlighted in blue and the path to reach them is indicated in red. The two strongly connected components are indicated with blue nodes connected by blue edges.

both the configuration with all the literals in the component assigned to 0 and all of them to 1 are present in at least one optimal solution. Indeed, an edge $(u, \overline{v})$ corresponds to a $u\overline{v}$ term in the posiform, which is null if the literals in the component have the same value. Moreover, based on the definition of the implication network and considering that in a strong component each node has a closed cycle starting and ending with itself, if a node is assigned a value of 1, all other nodes in that component can be reached from it and must also be assigned a value of 1. Alternatively, if a node is assigned to 0, all the nodes that can reach it, i.e., all the nodes in the strong component, must be assigned to 0.

If an edge exists between K_i and K_i', all the involved variables are strong persistencies. If the edge starts from a node in K_i, all literals in K_i must be assigned a value of 0, and all literals in K_i' must be assigned a value of 1 since they are complementary. Specifically, an edge starting from a node assigned 1 can only lead to another node assigned 1. Given that K_i and K_i' have complementary values by definition, all terms can vanish if and only if the starting node's value is 0 and the ending node's value is 1. Conversely, if the edge starts in K_i' and ends in K_i, all literals in K_i' are assigned a value of 0, and all literals in K_i are assigned a value of 1.

Example. *Continuing from the example in Section 4.1.1, after computing the maximum flow and obtaining the residual network it is possible to identify the two types of persistencies presented in this section (**Figure 11**).*

To detect this type of persistency, first of all, Tarjan's algorithm[42] is exploited to find all the strong components in the residual network. Successively, all the strong persistencies found can be substituted by their optimal values and removed from the residual network.

On the contrary, assigning weak persistencies is more complex because each is valid only in a subset of optimal solutions. Therefore, the assignments chosen for two distinct sets of weak persistencies may belong to different solutions. When combined, these assignments could yield a suboptimal solution. To evaluate if it is possible to assign them, it is necessary to verify if the component of weak persistencies can not be reached by any path originating from other persistencies. Indeed, in an implication network, two dependent variables are always connected by a path, and consequently, if a path connecting them does not exist, they are independent. Hence, if a component of weak persistencies can not be reached by any path originating from other persistencies, they can be indiscriminately assigned to 0 or 1. If other weak persistencies reach the component, these persistencies have to be assigned first, hence variables in the component are temporarily not assigned. To verify if other components of weak persistencies reach the current component, a reverse BFS can be utilized, which traverses the edges in the opposite direction. Subsequently, when a component, on which other groups of persistencies are dependent, is examined, a BFS is executed to find all variables it reaches. According to the implication network rules, where nodes assigned a value of 1 only lead to other nodes assigned to 1, an assignment is determined for all these variables.

4.2. Probing

The **Probing** technique aims to discover new persistencies and establish a lower bound for the function's minimum, which is based on the **roof dual bound** presented in ref. [35]:

$$c_0 + v(\varphi) \leq \min\phi(x) \tag{40}$$

where $\phi(x)$ indicates the posiform, c_0 is the constant term of the posiform and $v(\varphi)$ is the value of the maximum flow, defined as the maximum amount of flow exiting from the source node. Probing consists of generating two different networks for each variable x_i out of the n variables. The first receives an added penalty term for $x_i = 0$ and the second a penalty for $x_i = 1$ by inserting two new edges in the original residual network. Calling M the penalty coefficient, the following equation holds:

$$\min\phi(x) = \min(\min\phi(x) + Mx_i,\ \min\phi(x) + M\overline{x_i}) \tag{41}$$

To obtain the best roof dual estimation, M must be large enough to ensure that the maximum amount of flow is always computed. Knowing an upper bound U on the minimum of the posiform, if $M > U - c_0$, the edges exiting the source and entering the sink are able to carry more flow than the maximum, thus the largest amount of flow is calculated, and the roof dual bound achieves an approximation closer to the actual minimum. In the toolchain, Probing is directly applied to the residual network for which the constant term has already been dropped, hence $c_0 = 0$. The **Devour Digest Tidy-up (DDT)**[43] heuristic algorithm is used to obtain the upper bound U, in its one-pass version[44] to attain the result with the fastest implementation.

The Probing procedure is repeated for each variable. Therefore, computing $2N$ times the maximum flow. Calling LB_u the

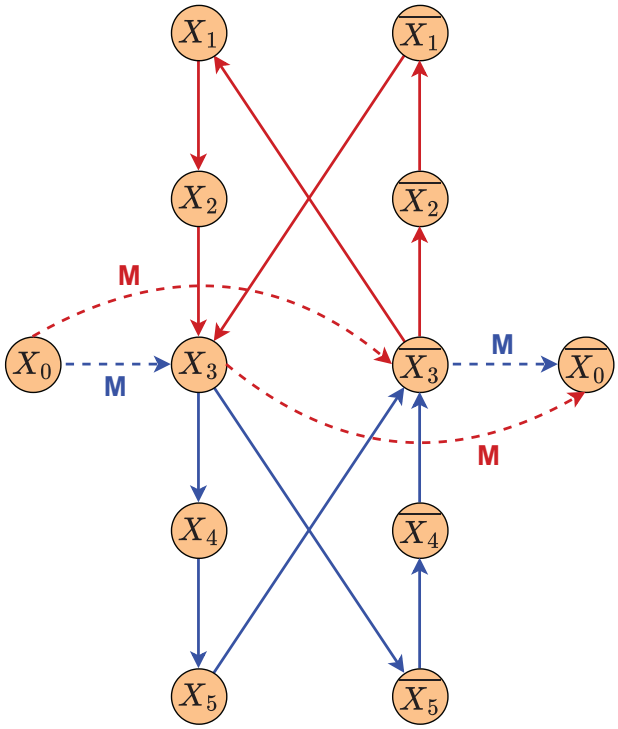


Figure 12. Example of Probing. The paths from source to sink are highlighted respectively in blue (penalty added for $\overline{x_3}$) and red (penalty for x_3).

lower bound obtained for the posiform $\phi(x) + M\overline{u}$ and $LB_{\overline{u}}$ that associated with the posiform $\phi(x) + Mu$, the lower bound of the original posiform can be improved in the following way:

$$LB = \max_{u \in L} \left(\min(LB_u, LB_{\overline{u}})\right) \tag{42}$$

where L is the set of literals. For any variable x_i, if the flow increases in both cases, the lower bound increases regardless of the value of x_i (**Figure 12**). The same method employed to find the lower bound can be applied to find the upper bound just by negating the QUBO cost function, i.e., pretending to maximize it instead of minimizing it when it is not trivial. In some cases, such as a max cut on a graph with only positive edges, the upper bound is found by assigning all the variables to 0 since the cut is null if no subset is identified. Thanks to the Probing technique, it is possible to estimate the range of values assumed by the QUBO function, which is fundamental to choosing a proper number of qubits for the quantum dictionary value in GAS and so for the algorithm effectiveness, as explained in Section 3.1.

Example. *From the residual network reported in Figure 12, it is possible to observe a successful application of Probing. If the variable x_3 is selected, a path from source to sink is formed when a penalty is added both for x_3 and $\overline{x_3}$. Consequently, the flow can be recomputed and the lower bound can be improved.*

Furthermore, the residual networks modified by Probing allow the detection of new persistencies. Since it is not known a priori the value of the variable chosen for Probing, the information of both networks must be combined to reduce the QUBO size further. Calling S_u and W_u the sets of strong and weak persistencies and L_u the lower bound identified for the posiform $\phi + M\overline{u}$:[35]

1. If $L_u > U$, $u = 0$ is a strong persistency for Φ;
2. if $v \in S_u \cap S_{\overline{u}}$, v is a strong persistency for Φ;
3. if $v \in W_u \cap W_{\overline{u}}$, v is a weak persistency for Φ;
4. if $v \in S_u$ and $\overline{v} \in S_{\overline{u}}$ then $u = v$ is a strong persistency for Φ;
5. if $v \in W_u$ and $\overline{v} \in W_{\overline{u}}$ then $u = v$ is a weak persistency for Φ;
6. for all $v \in S_u$ and $w \in S_{\overline{u}}$ the quadratic relations $u \leq v$, $\overline{u} \leq w$ and $\overline{w} \leq v$ are all strong persistencies for Φ;
7. for all $v \in W_u$ and $w \in W_{\overline{u}}$ the quadratic relations $u \leq v$, $\overline{u} \leq w$ and $\overline{w} \leq v$ are all weak persistencies for Φ.

With Probing, the discovered weak persistencies can be assigned and replaced in the network, as for the strong persistencies case. In a single iteration, all the weak persistencies are determined through the method described in Section 4.1, hence they belong to the same minimum and can be fixed. Across different iterations, the weak persistencies are mutually independent, allowing their identification and assignment. If this were not the case, there would have been an implication in the network, i.e. a path connecting the preceding persistency with the new one, allowing its detection. In essence, each persistency is independent of the previous one, otherwise, they would have been found simultaneously. Persistencies that do not find an assignment but a relation between variables (points from (4) to (7)) still have to be implemented in Qoolchain and will be the subject of future work.

4.3. Decomposition

The goal of decomposition is subdividing the problem into independent sub-instances with smaller sizes, which can be optimized in parallel, to reduce the overall execution time and also allow the usage of current quantum hardware.

4.3.1. Trivial Decomposition

The first method identifies mutually independent subfunctions in the residual network. If two subfunctions are defined on disjoint sets of variables, then:

$$f(x) = g(x) + h(x) \Rightarrow \min f(x) = \min g(x) + \min h(x) \tag{43}$$

This method is called **Trivial decomposition** and finds disconnected components in the residual network through the disjoint-set union-based algorithm. Due to symmetry, the corresponding functions act on disjoint sets of variables if there is no connection between two subnetworks.

4.3.2. Complete Strongly Connected Components

Once persistencies have been removed from the residual network, the cost function represented is homogeneously quadratic since all the linear terms vanished. When also all the non-complete strongly connected components are extracted, the only subgraphs left to examine in the network are the complete ones

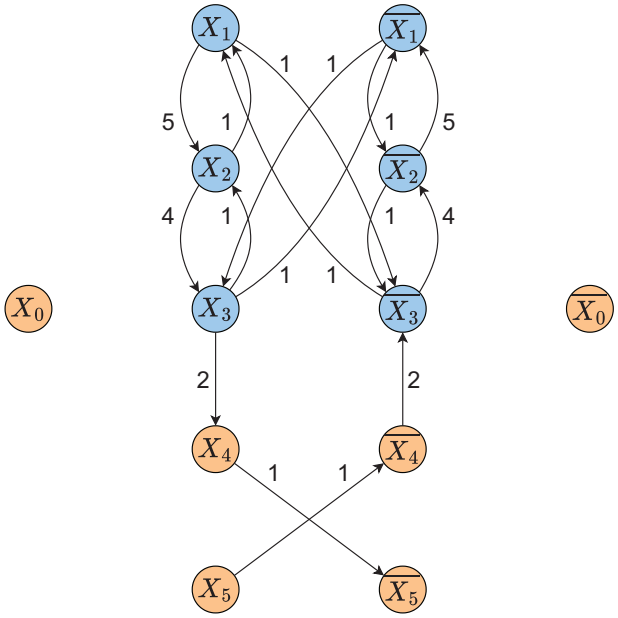


Figure 13. Example of a CSCC in a residual network (nodes highlighted in blue).

(**CSCCs**, **Figure 13**)—whose definition is provided in Section 4.1. Aspvall, Plass and Tarjan demonstrated in ref. [37] that conjunction of boolean clauses, where each clause is a disjunction of at most two literals can be satisfied if and only if there are no CSCCs in the implication graph associated with the boolean expression of these clauses. A conjunction of boolean clauses is expressed as $uv + wz$, while a term in a purely quadratic posiform takes the form $a_{uv}uv$. The unique distinction lies in the presence of weights, but the theorem is equally applicable. The theorem can be demonstrated by recalling that two conditions must occur for all the terms of the posiform to vanish:

- a literal x_i and its complement $\overline{x_i}$ have complementary values;
- no edge starts from a true node (assigned to 1) and ends in a false node (assigned to 0).

If a node u is in the same strongly connected component of its negated counterpart $\overline{u}$, any truth assignment to the graph nodes violates one of the two conditions mentioned above. Since there is a path from u to $\overline{u}$, either the two nodes have complementary truth values, or there is a path from a true node to a false one. Therefore, all the terms in a posiform can vanish if and only if no CSCC exists. If a true node leads only to true nodes and a false node is reached only by false nodes, thus, if no CSCC is present, the minimum of the corresponding posiform is 0. This is demonstrated in the following. Since CSCCs are subsets of the whole residual network it is known that:

$$\sum_{i=1}^{c} \min\Phi_i \leq \min\Phi \tag{44}$$

where c indicates the number of CSCCs in the graph and Φ_i is the posiform associated with the i^{th} component. The minimum of a posiform linked to a CSCC in the network is given by the sum of the smallest weights that, if removed, would make the component no longer strongly connected.[19] This value can be obtained by solving the QUBO problem corresponding to this CSCC. Therefore, a posiform generating a network with CSCCs can be considered as the sum of a CSCC-free posiform and T_i terms whose corresponding edges would produce c CSCCs, attaining that:

$$\min\Phi \leq \sum_{i=1}^{c} \sum_{k\in T_i} a_k = \sum_{i=1}^{c} \min\Phi_i \tag{45}$$

where a_k are the coefficients of the added terms. Recalling Equation (44), this demonstrates that the minimum of a posiform is equal to the sum of the minima of the posiforms associated with the CSCCs in the residual network:

$$\min\Phi = \sum_{i=1}^{c} \min\Phi_i \tag{46}$$

This means that the CSCCs can be extracted from the network and their corresponding QUBO functions can be solved independently. Successively, the variable assignments found can be substituted in the network achieving a CSCC-free QUBO function whose minimum is just given by the posiform's offset. Furthermore, a Grover oracle can be designed to identify the inputs' configuration associated with the value of the minimum for CSCC-free functions, which is known a priori. This makes the Grover Adaptive Search unnecessary, as Grover's search alone is sufficient to solve the optimization problem. Leveraging a purely quantum algorithm leads to faster execution, eliminating the need for classical iterations.

Table 3. Optimal solutions for the original problem and the decomposed one.

Solution no.	f $(x_1, x_2, x_3, x_4, x_5)$	Φ_{CSCC} (x_1, x_2, x_3)	Φ_{noCSCC} (x_3, x_4, x_5)
1	(0, 0, 0, 0, 0)	(0, 0, 0)	(0, 0, 0)
2	(0, 0, 0, 0, 1)	(0, 0, 0)	(0, 0, 1)
3	(0, 0, 0, 1, 0)	(0, 0, 0)	(0, 1, 0)
4	(0, 0, 1, 1, 0)	(0, 0, 1)	(1, 1, 0)
5	(0, 1, 1, 1, 0)	(0, 1, 1)	(1, 1, 0)
6	(1, 1, 1, 1, 0)	(1, 1, 1)	(1, 1, 0)

Example. *Considering the following cost function:*

$$f(x) = 4 + 8x_1 + 10x_2 + 4x_3 - 12x_1x_2 + 4x_1x_3 - 10x_2x_3 - 4x_3x_4 + 2x_4x_5 \tag{47}$$

*The residual network obtained by constructing the implication network and calculating the maximum flow is represented in Figure 13. The corresponding optimization problem has six different optimal solutions (**Table 3**). If the CSCC is extracted from the network, then*

two subnetworks are obtained, which are related to the two following posiforms:

$$\Phi_{CSCC}(x) = 10x_1\overline{x_2} + 2\overline{x_1}x_2 + 2x_1x_3 + 2\overline{x_1x_3} + 8x_2\overline{x_3} + 2\overline{x_2}x_3 \,. \quad (48)$$
$$\Phi_{noCSCC} = 4x_3\overline{x_4} + 2x_4x_5$$

In the CSCC-free posiform, all the nodes belonging to the CSCC and sharing an edge with a node outside the CSCC are also present in the function. When merged, the solutions of these two posiforms form the same solutions of the original optimization problem (Table 3).

4.3.3. Shannon Decomposition

After CSCC decomposition, the obtained subfunctions may still have a size larger than what current quantum hardware can solve. Hence, the goal is to apply a decomposition method that is always applicable and capable of breaking CSCCs for which, to the best of our knowledge, no technique is known so far. Therefore, in this article, we propose a new decomposition method called the **Shannon decomposition** since it applies the same principle of Shannon's expansion theorem for boolean algebra. The Shannon decomposition selects a variable x_i and evaluates the cost function for $x_i = 0$ and $x_i = 1$, producing two different subfunctions with one less variable. If applied recursively for all the variables, it is equivalent to solving an NP problem with a brute-force approach, which is highly inefficient. However, if the selected variable in a residual network with CSCCs is a **strong articulation point**, the two subfunctions' corresponding networks either have two CSCCs or none.

Definition. *A **strong articulation point**[45] is a node belonging to a strong component that if removed would make the component no longer strongly connected.*

Therefore, if two CSCCs are present in the achieved subnetworks, they can be split by the previously mentioned decomposition. If no CSCC is present, persistency-finding methods can be employed to further reduce the subfunction size. In conclusion, the subfunctions can be solved separately, and the optimal solution is determined by the subfunction with the lowest minimum value, combined with the assignments set on the variables chosen for the decomposition. Currently, the policy used to find the variable that most likely is an articulation point is evaluating the maximum degree in the residual network, i.e., the variable with the largest number of connections. This way, the greatest number of edges (or terms, from the associated posiform perspective) can be eliminated. The more edges are removed, the higher the probability of breaking a CSCC. This policy choice is not always optimal because an articulation point is not necessarily the highest degree node. Indeed, a better policy to directly detect such a point will be investigated in the future. Finally, the Shannon decomposition is applied a predetermined amount of times to limit its execution time, ensuring that its high computational complexity does not cancel the benefits of using only polynomial complexity algorithms.

Example. *Considering the QUBO cost function:*

$$f(x) = 12 - 4x_3 + 4x_1x_2 - 4x_1x_4 + 4x_2x_3 - 4x_2x_4 + 4x_3x_4 \quad (49)$$

*After performing all the toolchain steps described in the previous sections, the residual network at the top of **Figure 14** is obtained. This graph is composed of a whole strongly connected component. Now, let us apply Shannon decomposition and select the expansion variable as the one associated with the node with the highest degree. This approach ensures that the maximum number of terms, or edges, are removed after the function evaluation. Therefore, choosing x_2, the following two subfunctions are obtained:*

$$f_{\overline{x_2}} = 12 - 4x_3 - 4x_1x_4 + 4x_3x_4 \,. \quad (50)$$
$$f_{x_2} = 12 + 4x_1 - 4x_4 - 4x_1x_4 + 4x_3x_4$$

The corresponding residual networks are reported in Figure 14 on the left and the right, respectively. Noticeably, both networks do not have any CSCCs, the first one can be solved just by applying Grover's search and in the second all the variables are persistencies.

5. Results

5.1. Setup

All the tests were performed by exploiting the `Cython` implementation of the toolchain and the results were compared against those obtained with the D-wave preprocessing toolchain. All the benchmark problems were written with the qubovert[26] `Python` library and are available in the GitHub repository.[46]

The benchmarks considered are those described in Section 2.2.1. In addition, two well-known benchmarks were used for the max cut (*Gset*[47]) and graph coloring problem (*FullIns* and *Myciel*[48,49]).

Tests were executed on a single-process Intel(R) Xeon(R) Gold 6134 CPU @ 3.20 GHz opta-core, Model 85, with a memory of about 103 GB.[50]

5.2. Figures of Merit

This section presents all the metrics employed to assess the toolchain algorithms' efficacy and benchmark the results against other state-of-the-art techniques. Each figure of merit is designed to capture the effectiveness of each method based on the attributes of the problem under consideration. Below, the key figures of merit to assess the quality of the results are outlined:

- **Fixed persistencies percentage**: average fraction of the variables identified as persistencies, and thus removed from the QUBO function, over the total number of variables.
- **Number of strongly connected components decomposition**: number of times the decomposition techniques can divide the network into two subnetworks. If no decomposition is possible, the count is 0, if the graph is composed of a single CSCC and the remaining set of variables, the count is 1, thereafter, the count is equal to the number of CSCCs present in the residual network.
- **Total variable reduction**: average ratio, in percentage, between the size of the largest QUBO subproblem to be solved and that of the original problem. Alternatively, it can be defined as the average fraction of the variables that can be removed from the

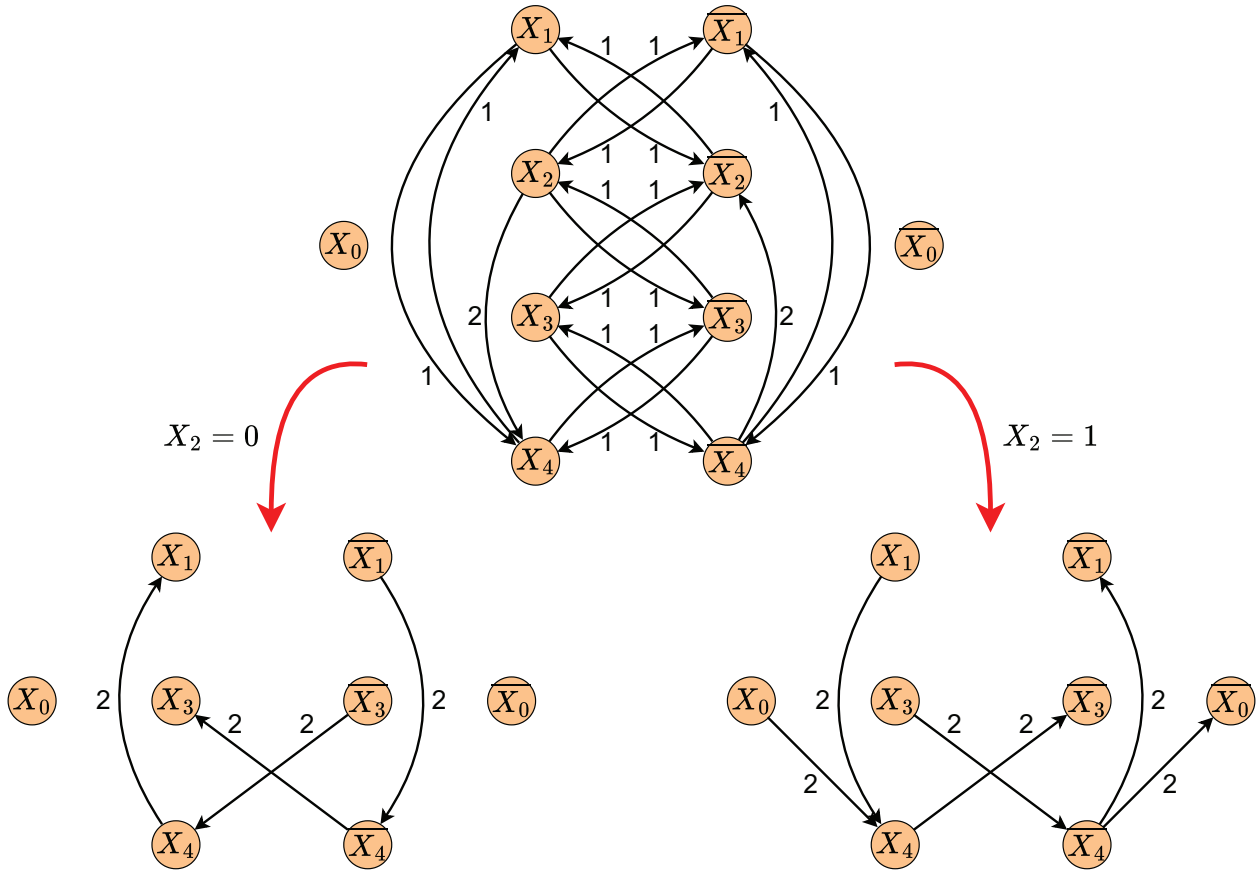


Figure 14. Shannon decomposition example. Starting from a strongly connected residual network, by selecting x_2 for the subfunctions evaluation, two subnetworks free of strongly connected components are generated.

QUBO function over the total number of variables. The removable variables are all the variables whose value can be detected a priori, through persistency-finding techniques and Probing, and all the variables, part of smaller subnetworks, pulled out by the decomposition techniques.

- **Error over the estimated range**: Relative error percentage comparing the estimated QUBO function range to the actual range or the range computed by a reference method.

$$e_r = \frac{|(UB - LB) - (UB^* - LB^*)|}{(UB^* - LB^*)} \tag{51}$$

where UB and LB respectively indicate upper and lower bound and the symbol * identifies the reference solution.

No metric has been defined for evaluating trivial decomposition results because this method rarely found two disconnected subsets in the residual networks. Consequently, no significantly sized disconnected components could be identified for all the benchmarks used.

5.3. Comparisons

This section presents the results of preprocessing with the toolchain and solving the benchmark problems outlined in Section 2.2.1. The evaluation involves comparing each method employed in the toolchain to one another and concerning other state-of-the-art algorithms, especially the D-wave preprocessing toolchain methods. For this purpose, the D-wave toolchain code was modified to output the number of persistencies fixed for each problem, allowing a direct comparison between this data and Qoolchain results.

Minimum vertex cover: For this benchmark, 1840 different graphs have been randomly generated. These instances range from a minimum of 10 nodes to a maximum of 100, with increments of 2, and with 4 density values of the corresponding QUBO formulation: 2%, 4%, 8%, 16%. For each size and density, 10 graphs are generated to calculate the results by averaging.

Figure 15a shows that the number of variables that can be removed from the QUBO cost function decreases by increasing the size and the density of the problem. For the 2% density problems, for around 100% of the variables its value can be deduced a priori, whereas for 16% density the number of variables removed is relevant only up to 60 nodes. By observing Figure 15b, the differences between this work and the D-wave toolchain are due to the Probing, decomposition, and Shannon techniques, which are not supported by the D-wave Toolchain. However, the improvements brought by Probing and decomposition do not provide a significant advantage, especially for larger densities. Furthermore, Figure 15c permits understanding why the CSCC de-

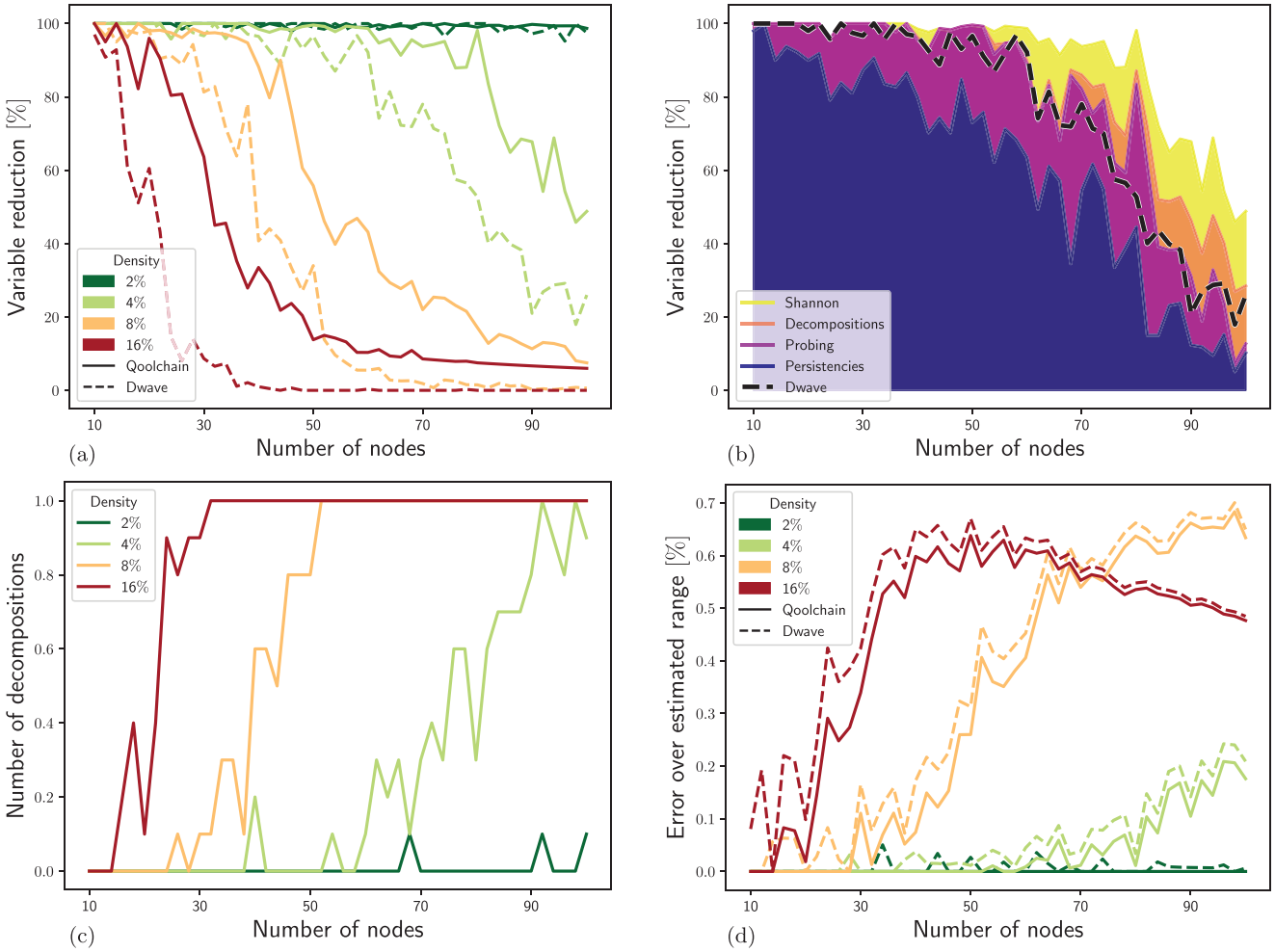


Figure 15. Qoolchain execution and comparisons for the generated minimum vertex cover problems. a) shows the average percentage of variable reduction comparing the results of this work with those of the D-wave toolchain. Up to 5 iterations of the Shannon decomposition have been employed in this case. b) highlights, in particular for problems with 4% density, the percentage of variables that can be removed with each of the methods employed by Qoolchain. The black dashed line indicates the results obtained with the D-wave toolchain. c) displays the average number of CSCCs identified in the residual networks corresponding to each problem. d) compares the error over the estimated range attained with Qoolchain and the D-wave toolchain.

composition gives a low advantage and persistency-finding techniques become less powerful increasing size and density. Following an opposite trend compared to what is observed in Figure 15a, the number of CSCC approaches 1 as size and density increase. This implies that persistency techniques and Probing manage to find a possible assignment for almost all the variables outside of a CSCC. Nevertheless, when a CSCC is encountered, no assignment can be determined. In addition, the maximum number of CSCCs found never exceeded 1, meaning that decomposition does not have the opportunity to split the problem into many smaller ones. It mainly separates a single CSCC from the rest of the network, which explains the limited advantages observed. This result highlights the primary issue that needs to be addressed. Improving persistency-finding techniques would not significantly reduce the number of variables, as they are likely located within a single CSCC. The meaningful reduction in problem size would come from breaking the CSCC into multiple smaller graphs. This is the goal of the proposed Shannon decomposition, which may not substantially impact this benchmark because persistencies are easier to identify compared to other benchmarks.

Finally, Figure 15d showcases that Probing enhances the accuracy of the estimated range of the cost function. However, for this benchmark, where many variables have already been removed using the previously discussed techniques, the error achieved by the D-Wave toolchain, employing only roof duality, is already minimal. This allows for an extremely precise prediction of the function's bounds.

Maximum clique: As for the previous case, 1840 different graphs have been randomly generated. These instances range from a minimum of 10 nodes to a maximum of 100, with increments of 2, and with 4 density values of the corresponding QUBO formulation: 2%, 5%, 10%, 20%. For each size and density, 10 graphs are generated to calculate the results by averaging.

The results in **Figure 16** are close to those obtained with minimum vertex cover, coherently with expectations since the minimum vertex cover and the maximum clique are complementary problems[51] (given a graph G, a subgraph is a clique if and only if

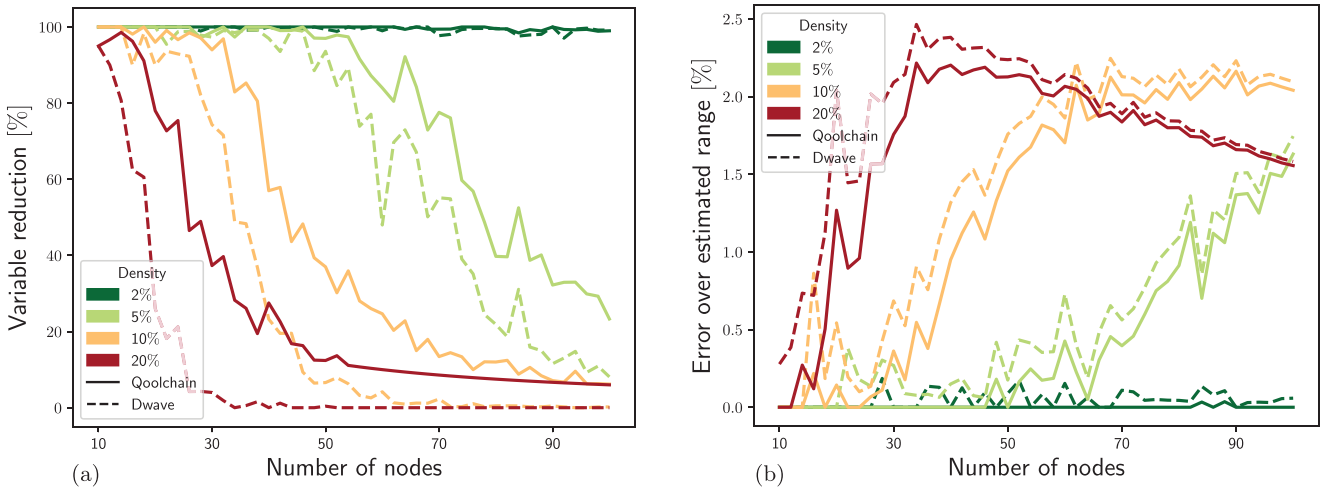


Figure 16. Qoolchain execution and comparisons for the generated maximum clique problems. The results for the average percentage of variable reduction Panel (a) and the error over the estimated range Panel (b) compared with the D-wave implementation are consistent with those of minimum vertex cover results.

all the nodes outside it form a vertex cover in the complement of G). The trends observed are analogous also in comparison with the D-wave toolchain.

Number partitioning: 640 different instances of the number partitioning problem have been randomly generated, with sets sizes ranging from a minimum of 10 numbers to a maximum of 40, with increments of 2. Numbers have been extracted from 4 different sets of integers starting from 1 and having as largest number 3, 5, 15, and 20. 10 instances are generated for each set size and largest number to calculate the results by averaging.

Every number partitioning problem generated has exactly one CSCC. This makes this benchmark particularly challenging for the toolchain's techniques. For both Qoolchain and D-wave cases, no variable can be eliminated from the QUBO cost function. Even with smaller values for the largest number in the set, the densities of the QUBO matrices are 100%, resulting in the residual network consisting of a single dense CSCC, making the algorithms ineffective. **This confirms the hypothesis that variable values can be detected a priori only if they are not part of a CSCC**. In these cases, the CSCC is so dense, even for smaller sets, that the Shannon decomposition does not allow the removal of more variables than that selected for the decomposition. Even without persistency identification, Probing is more accurate than the roof-duality alone for the estimation of the function range, as shown in **Figure** 17b, reporting the comparison between Qoolchain and the D-wave toolchain. In this benchmark, the benefits of the probing technique are more relevant than in the others.

Max cut: 1040 different graphs have been randomly generated. These instances range from a minimum of 10 nodes to a maxi-

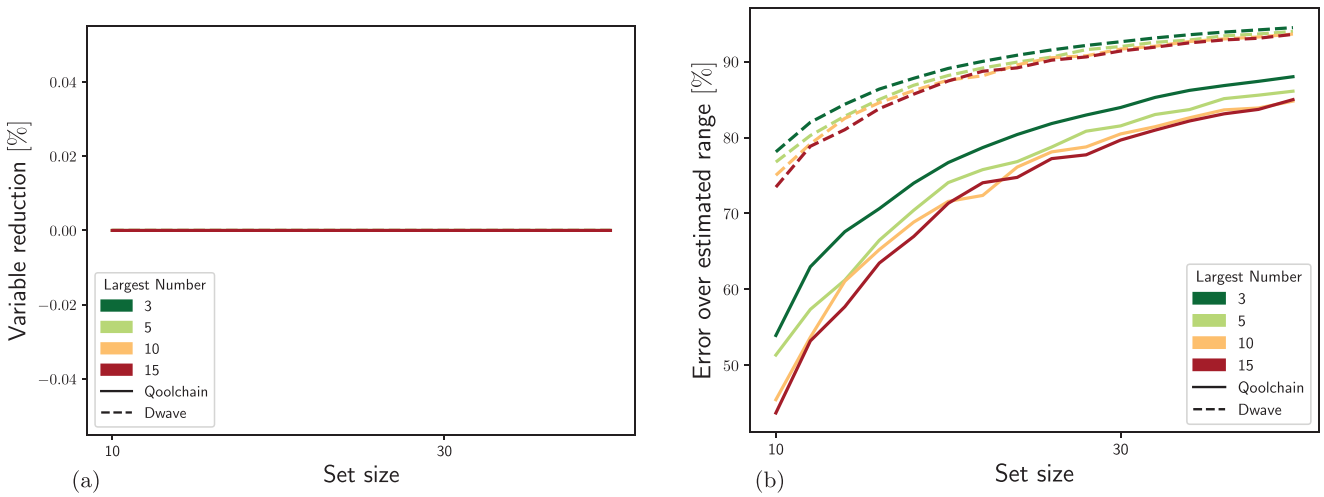


Figure 17. Qoolchain execution and comparison for the generated number partitioning problems. In both this work and the D-Wave toolchain, no variables can be removed from the cost function (a). However, the additional techniques in Qoolchain result in a more accurate estimation of the function range, particularly when compared to problems with smaller CSCCs, as opposed to the results achieved by the D-Wave toolchain (b).

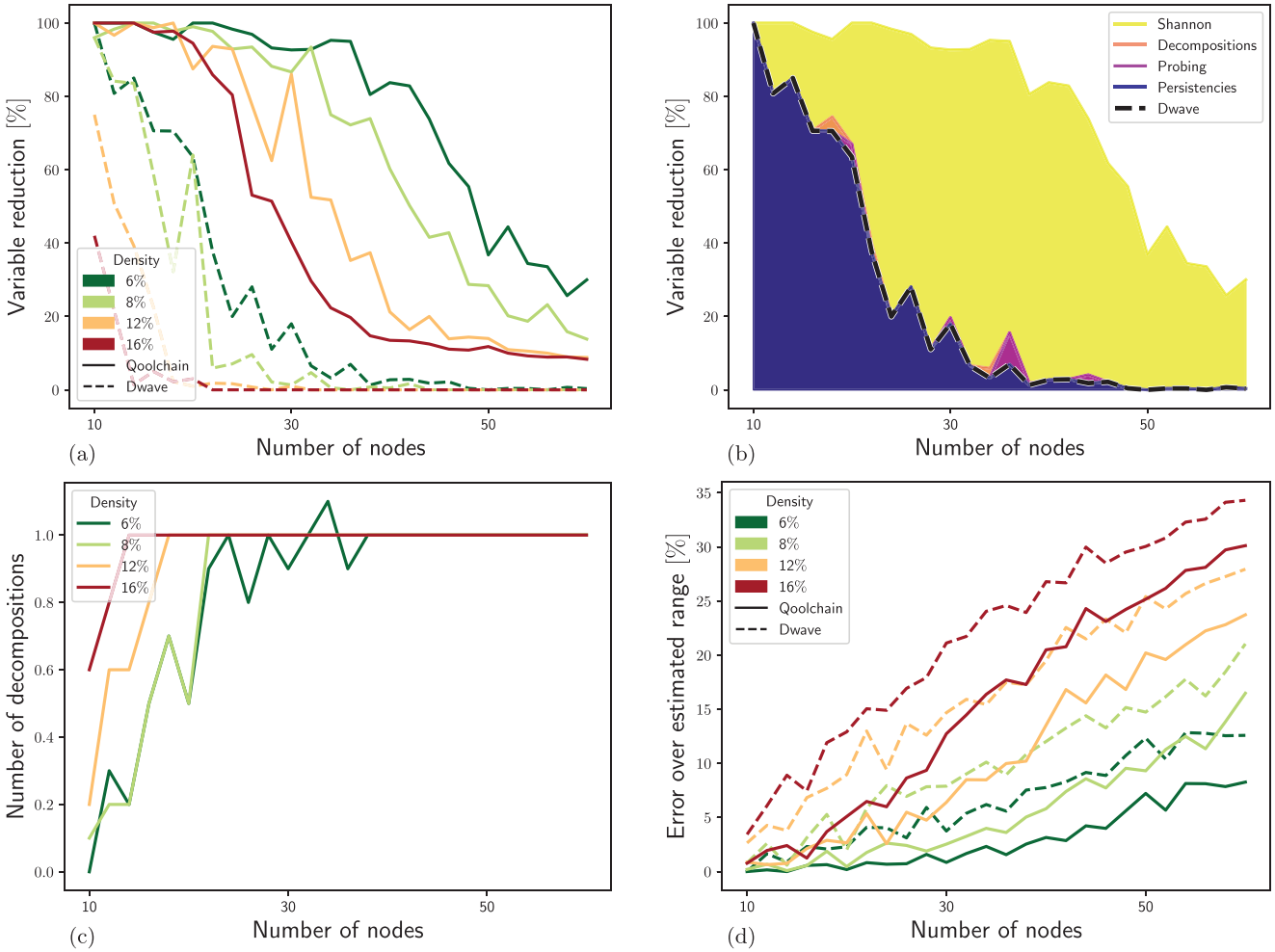


Figure 18. Qoolchain execution and comparisons for the generated max cut problems. Panel (a) shows that this benchmark is more challenging from the variable reduction point of view compared to the others, even if the advantage of Qoolchain techniques over D-wave toolchain ones is more evident. Also in this case, up to 5 iterations of the Shannon decomposition have been employed. Panel (b) details the percentage of variables that can be removed with each of the methods employed by Qoolchain for problems with 6% density. Notably, the Shannon decomposition carries a meaningful advantage compared to the D-wave toolchain, represented by the dashed line. Panel (c) displays that the average number of CSCCs is always around 1 over a certain graph size. d) Concerning the error over the estimated range, residual networks for this benchmark are formed of denser CSCC, and following the trend observed with number partitioning, Probing improves the effectiveness of the toolchain.

mum of 60, with increments of 2, and with 4 density values of the corresponding QUBO formulation: 6%, 8%, 12%, 16%. 10 graphs are generated for each size and density to calculate the results by averaging.

Coherently with previous results, max cut residual networks have a structure that is more likely to comprise a single CSCC than minimum vertex cover, but less than number partitioning. Indeed, results for all metrics (**Figure 18**) are between the other two benchmarks. Anyway, what stands out in these results is that the Shannon decomposition is particularly effective for max cut. Notably, for all densities, the Shannon decomposition is particularly effective for max cut problems compared to that of the other benchmarks and the improvement over the D-wave toolchain is greater for max cut problems than for the other benchmarks (Figure 18a). Figure 18b shows that Probing and decomposition are poorly effective, and the overall improvement compared to the D-Wave toolchain is mostly due to the Shannon decomposition. For graph sizes up to around 40 nodes, just by selecting and fixing the value of 5 variables, it is possible to find a priori the value of almost 100% of the variables. This proves that, under certain circumstances, it is possible to break CSCCs and this is the only way to reduce the dimensionality of the related QUBO formulation for such complex problems. Unfortunately, this is less likely to happen for larger graphs. Therefore, a greater number of iterations and especially an improved variable selection policy could enhance the outcomes.

Furthermore, Qoolchain has been executed on the *Gset*,[47] a renowned max cut benchmark widely used for testing and validating combinatorial optimization algorithms, which involves only graphs having 1 or −1 as edges' weights.

Gset graphs start from a minimum of 800 nodes, hence all the graphs are entirely constituted of a CSCC and neither Qoolchain nor the D-wave toolchain can eliminate any variable from the original QUBO formulation. However, estimating the range of

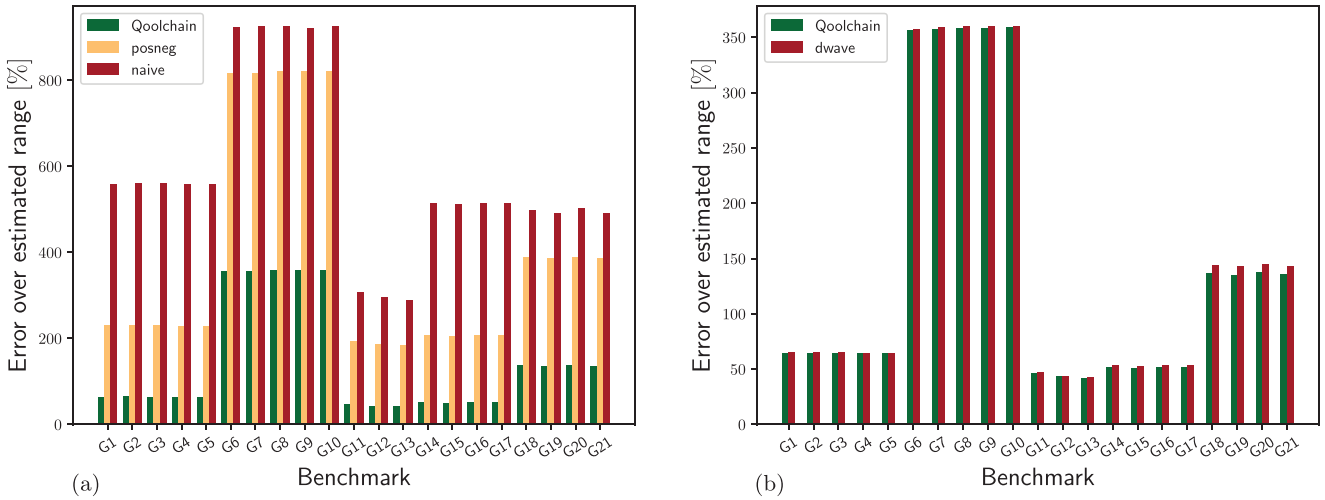


Figure 19. Qoolchain execution and comparisons for the Gset benchmark. Panel (a) reports the error over the estimated range compared with the *naive* and *posneg* methods, whereas Panel (b) shows the comparison with the D-wave implementation.

values that the function assumes is still possible. This metric is obtained with the D-wave toolchain and two greedy methods (*Naive* and *Posneg*) described in ref. [23, 52]. The latter methods were originally conceived as bounds estimations to calculate QUBO penalty coefficients, but they are still valid for estimating an upper bound on the function range. Comparisons are reported in **Figure 19**. Naive and Posneg are one-pass methods, making them the fastest; nevertheless, their estimation errors are, at best, more than double those achieved by Qoolchain. This work also performs better than the D-Wave toolchain for all Gset problems tested. In this case, unlike the randomly generated max cut instances, the minimal improvement does not justify the higher computational complexity related to the Probing technique application for this benchmark problem, for such large graph sizes. To give an idea about the quality of the estimation, the best-known value for the QUBO function minimum of the *G1* problem is −11624,[53] while the maximum is 0, equivalent to finding no cut. Therefore, 14 qubits ($\lceil \log_2(|-11624|) \rceil = 14$) are necessary to encode the function values in a quantum state. Qoolchain's estimation is −19142, hence just one more qubit is needed ($\lceil \log_2(|-19142|) \rceil = 15$).

Graph coloring: The graph coloring problem requires a lower bound on the chromatic number to define a sufficient number of variables in the QUBO formulation. Two online benchmarks (*FullIns* and *Myciel*[48,49]), for which the chromatic number is known, have been employed to test Qoolchain and compare it with state-of-the-art algorithms.

Figure 20 shows similar results to the Gset tests. In this case as well, the graphs are entirely constituted of a CSCC, hence no variable can be removed from the QUBO formulation. Despite being the fastest, the *Posneg* and *Naive* methods achieve a worse range estimation, at best double that of this work. Moreover, for

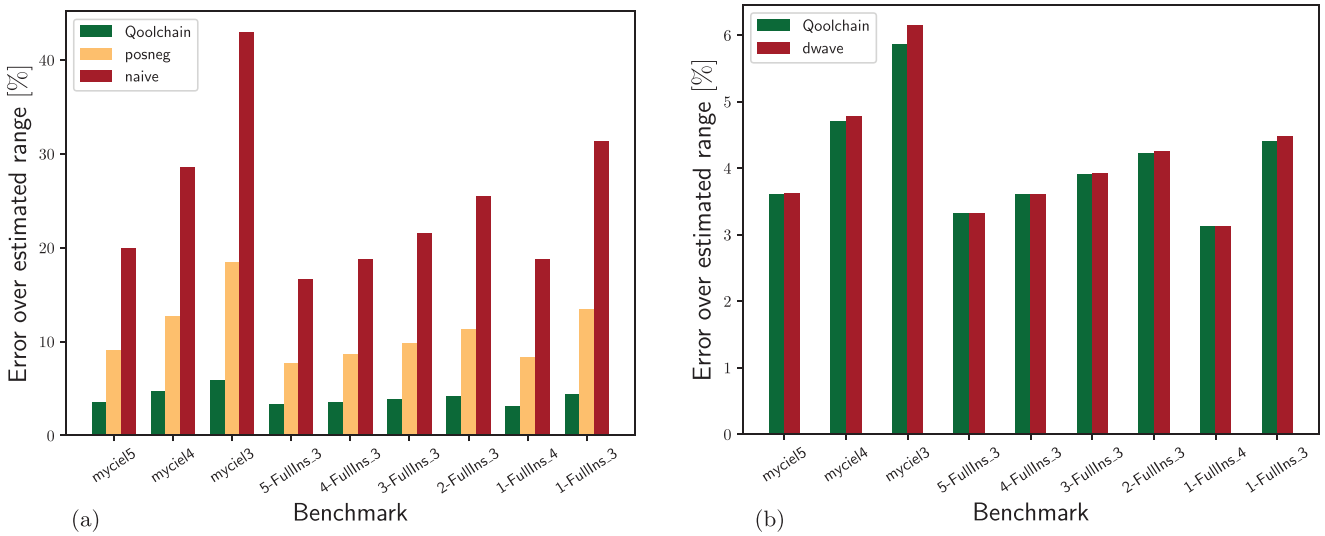


Figure 20. Qoolchain execution and comparisons for *Myciel* and *FullIns* graph coloring benchmarks. The error over the estimated range calculated with Qoolchain is compared with the *naive* and *posneg* methods (a) and the D-wave implementation (b).

all the problem instances Qoolchain performs better than the D-wave toolchain. Nevertheless, the improvements are not significant enough to justify a substantial increase in computational cost related to the Probing technique application for this benchmark problem.

In this case, to reduce the number of involved variables, it is more advantageous to use the alternative formulation of the problem proposed in [15], particularly when a quantum solver capable of handling PUBO problems, such as GAS, is considered.

6. Conclusion

This work introduces Qoolchain, a QUBO preprocessing toolchain designed to reduce problem complexity and enhance the exploitability of quantum solvers with current quantum computers. Developed in `Cython` for compatibility with major quantum frameworks and leveraging the efficiency of C++, Qoolchain is publicly available on GitHub.[14] It is compatible with any QUBO-compliant solver but is specifically optimized for GAS.

The toolchain consists of several steps, each with at most polynomial complexity. These steps include persistency identification and problem decomposition to reduce the number of variables, as well as the Probing technique to estimate the upper and lower bounds of the problem's cost function. Additionally, this work proposes using the Grover algorithm independently to address segments of the decomposed problem whose minimum value is known in advance, thereby reducing the time required for solving the problem. The Shannon decomposition method is also introduced. Although it has exponential complexity, it can effectively break the CSCC of the residual network associated with the QUBO problem, permitting a further reduction in complexity.

The effectiveness of Qoolchain is evaluated against the D-Wave preprocessing toolchain across several problems: Minimum Vertex Cover, Maximum Clique, Number Partitioning, Max Cut, and Graph Coloring. For all the problems considered, our tool demonstrates higher efficiency than the D-Wave preprocessing toolchain in reducing the complexity of the QUBO to solve. It also shows greater accuracy compared to both the D-Wave preprocessing toolchain and other methods implemented in the literature (such as *Naive* and *Posneg*) in estimating function bounds.

Furthermore, the results reveal that the effectiveness of QUBO reduction strategies depends on specific problem characteristics, such as the density of the associated QUBO matrix. For example, variable reduction techniques are highly effective for Minimum Vertex Cover, Maximum Clique, and Max Cut problems but are ineffective for Number Partitioning and Graph Coloring. This inefficacy is due to Number Partitioning and Graph Coloring being a high-density problem comprised of a single CSCC, which prevents persistency identification or problem decomposition.

Additionally, our results highlight that the proposed Shannon decomposition is particularly efficient for the Max Cut problem, as it permits breaking the CSCC and identifying further persistency through repeated application of the technique.

Even though the results are promising and demonstrate the potential of the toolchain, there are several areas for improvement. Integrating methods for handling quadratic persistency could enhance its QUBO reduction capabilities. Additionally, implementing a coefficient normalization step to limit the number of qubits required for value representation can significantly benefit the GAS. Furthermore, more rigorous and efficient techniques for node selection in Shannon decomposition could improve the chance of breaking the CSCC. Finally, introducing solver-dependent steps tailored to optimize preprocessing for specific solvers, such as QA, QAOA, and VQE, could enhance overall performance.

Moreover, an analysis of the performance on problems with natively higher-order terms reduced to quadratic form would be valuable, as it could help identify situations where it is more advantageous to solve the PUBO problem directly—if the solver can handle it—or to leverage a combination of polynomial reduction and Qoolchain functions.

In conclusion, the proposed toolchain represents a significant step forward in enabling the exploitability of the quantum solver, taking into account the limitations of the current quantum hardware and permitting it to handle more complex problems relevant to industry scenarios.

Acknowledgements

Open access publishing facilitated by Politecnico di Torino, as part of the Wiley - CRUI-CARE agreement.

Conflict of Interest

The authors declare no conflict of interest.

Data Availability Statement

The data that support the findings of this study are available from the corresponding author upon reasonable request.

Keywords

grover adaptive search, grover search, preprocessing, quantum computing, quadratic unconstrained binary optimization

Received: August 7, 2024
Revised: November 12, 2024
Published online: December 13, 2024

[1] S. Bouajaja, N. Dridi, *Oper. Res.* **2017**, *17*, 339.
[2] B. Zhou, J. Bao, J. Li, Y. Lu, T. Liu, Q. Zhang, *Rob. Comput. Integr. Manuf.* **2021**, *71*, 102160.
[3] H. Zheng, Y. Feng, J. Tan, *IEEE Access* **2017**, *5*, 12648.
[4] Z. A. Abdalkareem, A. Amir, M. A. Al-Betar, P. Ekhan, A. I. Hammouri, *Health Technol* **2021**, *11*, 445.
[5] D. Henderson, S. H. Jacobson, A. W. Johnson, in *Handbook of Metaheuristics* (Eds.: F. Glover, G. A. Kochenberger), Springer US, Boston, MA **2003**, pp. 287–319.
[6] T. Kadowaki, H. Nishimori, *Phys. Rev. E* **1998**, *58*, 5355.
[7] B. K. Chakrabarti, H. Leschke, P. Ray, T. Shirai, S. Tanaka, *Philos. Trans. R. Soc., A* **2023**, *381*, 20210419.
[8] A. Rajak, S. Suzuki, A. Dutta, B. K. Chakrabarti, *Philos. Trans. R. Soc., A* **2023**, *381*, 20210417.
[9] K. Yukiyoshi, N. Ishikawa, *arXiv preprint arXiv:2211.04637* **2022**.

[10] E. Combarro, S. Gonzalez-Castillo, A. Di Meglio, *A practical guide to quantum machine learning and quantum optimization: Hands-on approach to modern quantum algorithms*, Packt Publishing, Birmingham, UK **2023**.
[11] M. A. Nielsen, I. L. Chuang, *Quantum computation and quantum information*, Cambridge University Press, Cambridge; New York, 10th anniversary ed edition, **2010**.
[12] L. Giuffrida, D. Volpe, G. A. Cirillo, M. Zamboni, G. Turvani, *IEEE J. Emerging Sel. Top. Circuits Syst.* **2022**.
[13] C. Durr, P. Hoyer, *arXiv:quant-ph/9607014* **1996**.
[14] A. Gilliam, S. Woerner, C. Gonciulea, *Quantum* **2021**, *5*, 428.
[15] Y. Sano, K. Mitarai, N. Yamamoto, N. Ishikawa, *IEEE Transactions on Quantum Engineering* **2024**.
[16] L. K. Grover, *arXiv:quant-ph/9605043* **1996**.
[17] A. Peruzzo, J. McClean, P. Shadbolt, M.-H. Yung, X.-Q. Zhou, P. J. Love, A. Aspuru-Guzik, J. L. O'Brien, *Nat. Commun.* **2014**.
[18] E. Farhi, J. Goldstone, S. Gutmann, *arXiv.1411.4028* **2014**.
[19] A. Billionnet, B. Jaumard, *Operations Research Letters* **1989**, *8*, 161.
[20] F. Glover, G. Kochenberger, Y. Du, *arxiv.1811.11538* **2018**.
[21] M. Anthony, E. Boros, Y. Crama, A. Gruber, *Math. Program.* **2017**.
[22] M. Ayodele, in *European Conference on Evolutionary Computation in Combinatorial Optimization (Part of EvoStar)*. Springer, Berlin, Germany **2022** 159–174.
[23] M. D. García, M. Ayodele, A. Moraglio, in *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. **2022** 184–187.
[24] A. Verma, M. Lewis, *Discrete Optim.* **2022**, *44*, 100594.
[25] K. Tanahashi, S. Takayanagi, T. Motohashi, S. Tanaka, *J. Phys. Soc. Jpn.* **2019**, *88*, 061010.
[26] qubovert Documentation, qubovert documentation - getting started, https://qubovert.readthedocs.io/en/latest/, (accessed: December 2023).
[27] C. Cook, H. Zhao, T. Sato, M. Hiromoto, S. X.-D. Tan, *Integration* **2019**, *69*, 335.
[28] Y. Haribara, S. Utsunomiya, K.-i. Kawarabayashi, Y. Yamamoto, *arXiv preprint arXiv:1501.07030* **2015**.
[29] F. Liers, T. Nieberg, G. Pardella, Via minimization in vlsi chip design application of a planar max-cut algorithm, **2011**, http://e-archive.informatik.uni-koeln.de/630/.
[30] M. Garey, D. Johnson, H. So, *IEEE Trans. Circuits Syst.* **1976**, *23*, 591.
[31] M. Norimoto, N. Ishikawa, in *2023 IEEE 97th Vehicular Technology Conference (VTC2023-Spring)*. **2023** 1–5.
[32] L. K. Grover, *Phys. Rev. Lett.* **1997**, *79*, 325.
[33] M. Norimoto, R. Mori, N. Ishikawa, *IEEE Trans. Commun.* **2023**, *71*, 1926.
[34] Dwave preprocessing toolchain, https://github.com/dwavesystems/dwave-preprocessing.
[35] E. Boros, P. Hammer, G. Tavares, *RUTCOR Research Report* **2006**.
[36] S. Behnel, R. Bradshaw, C. Citro, L. Dalcin, D. S. Seljebotn, K. Smith, *Comput. Sci. Eng.* **2011**, *13*, 31.
[37] B. Aspvall, M. F. Plass, R. E. Tarjan, *Inf. Process. Lett.* **1979**.
[38] B. Korte, J. Vygen, *Combinatorial Optimization: Theory and Algorithms*, Springer Berlin Heidelberg, Berlin, Heidelberg **2012**.
[39] A. V. Goldberg, R. E. Tarjan, *J. ACM* **1988**, *35*, 4.
[40] B. V. Cherkassky, A. V. Goldberg, *Algorithmica* **1997**.
[41] J. Cheriyan, S. N. Maheshwari, *SIAM J. Comput.* **1989**.
[42] R. Tarjan, *SIAM J. Comput.* **1972**.
[43] F. Glover, B. Alidaee, C. Rego, G. Kochenberger, *Eur. J. Oper. Res.* **2002**, *137*, 272.
[44] S. Hanafi, A.-R. Rebai, M. Vasquez, *J. Heuristics* **2013**.
[45] G. F. Italiano, L. Laura, F. Santaroni, *Theor. Comput. Sci.* **2012**, *447*, 74.
[46] Qoolchain github repository, https://github.com/ilRenato/Qoolchain.
[47] Gset, https://web.stanford.edu/~yyye/yyye/Gset/, (accessed: October 2024).
[48] Color02/03/04: Graph coloring and its generalizations, https://mat.tepper.cmu.edu/COLOR02/, (accessed: June 2024).
[49] Vertex coloring, https://sites.google.com/site/graphcoloring/vertex-coloring, (accessed: June 2024).
[50] Intel Xeon Gold 6134 processor - product specification, [Online] https://ark.intel.com/content/www/us/en/ark/products/120493/intel-xeon-gold-6134-processor-24-75m-cache-3-20-ghz.html, (accessed: October 2021).
[51] E. Pelofske, G. Hahn, H. Djidjev, *J. Signal Process. Syst.* **2021**.
[52] J. Pauckert, M. Ayodele, M. D. García, S. Georgescu, M. Parizy, in *Proceedings of the Companion Conference on Genetic and Evolutionary Computation*, GECCO '23 Companion. Association for Computing Machinery, New York, NY, USA **2023** 227–230.
[53] Y. Matsuda, Benchmarking the max-cut problem on the simulated bifurcation machine, **2019**, https://medium.com/toshiba-sbm/benchmarking-the-max-cut-problem-on-the-simulated-bifurcation-machine-e26e1127c0b0.