POLITECNICO DI TORINO

Master degree course in Electronic Engineering

Master Degree Thesis

QUBO models preprocessing toolchain for quantum solvers

Supervisors
Prof. Maurizio Zamboni
Prof.ssa Mariagrazia Graziano
Prof.ssa Giovanna Turvani

Candidate
Giacomo Orlandi

April 2023

Summary

Combinatorial optimization problems aim to find the configuration of inputs that minimizes a cost function and can be employed in many real-world applications. Most of them are known to be NP-complete or NP-hard, thus classically solvable with exponential complexity. Quadratic Unconstrained Binary Optimization (QUBO) formulation can express them in a format suitable for quantum solvers, which can reduce the computational complexity by exploiting the quantum principles. One of the most recent solvers, the Grover Adaptive Search (GAS), is a successive approximation method that iteratively shifts the cost function whenever a negative value is obtained until the minimum is achieved. It encodes the function in a quantum state as key-value pairs and exploits Grover's search (GS), a quantum routine able to explore an unordered dataset with lower complexity than the classical counterpart, for obtaining negative samples. Since current quantum hardware still has a limited number of qubits, this thesis seeks to build a QUBO preprocessing toolchain for quantum solvers, in particular for GAS algorithm, focused on reducing the number of variables of the problem. The toolchain uses Cython to bridge between Python, which is used by most quantum backend APIs, and C++, which is used to implement the algorithms guaranteeing the efficiency of a compiled language.

The toolchain consists of many steps of at most polynomial complexity. First, the cost function is transformed to an equivalent network form to find persistencies, i.e., variables to which binary values valid in at least one optimal solution can be assigned. Two routines exploit graph traversals and strongly connected components (SCCs), which are subgraphs where every node is reachable from any other node, to find these assignments. Finally, by eliminating the persistencies, a homogeneously quadratic function is obtained.

After these procedures, decomposition routines divide the network into smaller ones: separating the disconnected components, which correspond to independent subfunctions, and extracting the remaining SCCs. The latter operation is needed because the minimum of a purely quadratic function can be obtained just by finding the minima of the subfunctions associated with these components. In this way, when solving the networks with no SCCs the minimum is known in advance, thus applying GS instead of GAS is sufficient to obtain the optimal solution, which

significantly decreases the execution time. The presented mechanisms also permit the decomposition of specific classes of problems and the computation of a lower bound for the optimal solution, which allows the estimation of the number of qubits necessary for representing the cost function values in a quantum state.

Finally, if the available hardware resources are still insufficient, the Shannon decomposition partitions the problem into two functions: one in which a chosen variable is equal to 1 and the other in which it is 0. Besides, if the variable removal breaks an SCC into two, the subfunction can be in turn decomposed.

The results show that the persistency-finding techniques are very efficient in removing all the variables that are not part of SCCs. For many benchmarks, once persistencies are taken out, decompositions reveal that functions are composed of a unique SCC. Moreover, the SCC usually has a size not small enough to directly run GS on currently available backends. Hence, breaking down SCCs is the only way to furtherly reduce the number of variables.

In the first chapter, the work motivations are illustrated, and a brief introduction to the quantum computing paradigm, with a particular focus on quantum formalism and commonly employed algorithms, is reported.

The Quadratic Unconstrained Binary Optimization formalism is presented in the second. Moreover, the problems employed for benchmarking are showcased, explaining how it is possible to write them according to the QUBO formalism.

The Grover Adaptive search algorithm and its degrees of freedom are introduced in the third chapter.

The fourth chapter describes the structure of the proposed preprocessing toolchain, detailing the choice of the programming language and the included blocks.

The fifth chapter describes how to obtain the network representation of a QUBO problem for applying the preprocessing algorithms for variable reduction, problem decomposition, and estimation of the function range.

Persistencies are defined in the sixth chapter, and the algorithms for identifying them are explained. In addition, the probing technique, which permits more accurate estimation of the function range and identification of further persistency, is presented.

Furthermore, decomposition methods are explained in the seventh chapter. In particular, trivial, strong components, Shannon, and problem-specific decomposition approaches are considered.

The eighth chapter reports and comments on the obtained results, with particular attention on the benefit that can be reached in the exploitation of the Grover Adaptive Search solver.

Future perspectives are presented in the ninth chapter. In particular, possible approaches for toolchain expansion and improvement are suggested.

In the end, conclusions are drawn in the tenth chapter.

Contents

Part I

Theoretical foundations

Chapter 1

Introduction

1.1 Motivation

The **optimization** goal is identifying the configuration of input variables that minimizes or maximizes an **objective function** (**cost** or **fitness function** if referred to minimization or maximization). In the rest of this thesis, optimization will always be referred to as a minimization problem, considering that a maximization one can be written as a minimization by changing the sign of the objective function. In some applications, not all the input configurations are allowed, hence some constraints have to be applied to variables (constrained optimization). Optimization problems can be defined as **combinatorial** if they deal with discrete variables. They find applications in several fields, such as finance [1, 2], scheduling [3], VLSI design [4] and electoral legislation [5]. Such problems are NP-hard or NP-complete problems, thus they are classically solvable in a runtime scaling exponentially with their size. A **brute-force approach** computes the solution for every possible combination of the input variables configurations and it clearly grants to find the optimal one, but at the expense of an unfeasible computational cost even for problems of small dimensions. Other classical methods such as stochastic ones can achieve a reasonable runtime, but it is not guaranteed that the optimal solution will be found. One of the best-known algorithms belonging to this category is **simulated annealing** (**SA**) [6], which arises from an analogy with the thermal annealing process. At every iteration, it selects a new solution that is accepted if it improves the previous one. If it worsens the attained value, it is accepted with a probability depending on a parameter called temperature. The value of this parameter decreases as the algorithm proceeds. In this way, at the beginning of the process, when the probability to accept a worse solution is higher, it is easier to escape local minima. Real-world problems cost functions are typically multimodal, i.e., they present several local minima. These are values that are optimal in a neighborhood but are not the lowest value assumed by the whole function. However, all the classical methods may never converge, or take an unfeasible amount of time or achieve suboptimal

solutions. Therefore, the quantum properties could be exploited to offer new optimization methods having the potential to overcome critical issues of classical ones. Two quantum computing paradigms can be used in this context: **quantum annealing** and **gate-based computing**. The first, which is the quantum counterpart of simulated annealing, is based on an adiabatic evolution of a quantum system, which permits reaching the ground state of the problem energy profile, namely, the minimum of the cost curve. Thanks to quantum tunneling, it is able to overcome high and narrow barriers in the function avoiding getting stuck in local minima. It exploits a special-purpose computer since it is only used for optimization. The second paradigm, in which the evolution of a quantum state can be achieved through a sequence of unitary transformations called quantum gates, is based on the use of general-purpose quantum computers that can be used to implement optimization algorithms. Current quantum annealers or gate-based quantum computers can not directly compute the solution of any combinatorial optimization problem because they face a severe limitation on the available qubits [7], i.e., quantum bits. A higher amount of qubits causes greater noise, thus the state of the qubits would not be stable and it may lead to errors. For this reason, in current hardware, fewer qubits are available than many objective functions require.

This thesis aims to develop a preprocessing toolchain for improving the exploitability of quantum approaches in an optimization context, with a particular focus on a recently proposed algorithm, called Grover Adaptive Search. The goal is to decompose cost functions into multiple subfunctions with a lower number of variables. Reduction techniques can also be employed, alone or in addition to decomposition, to remove variables whose optimum value can be deduced a priori. Once the qubits demand of the reduced and decomposed function is compatible with available hardware, the minimization problems can be solved. Still, the newly obtained functions must ensure that the results can be combined to find the solution to the original problem.

1.2 Quantum computing

In order to provide sufficient knowledge of the theoretical concepts necessary for the comprehension of this thesis's work, the following section is an introduction to the quantum computing paradigm, which is the result of the study of Yanofsky and Mannucci's "Quantum computing for computer scientists" [8].

1.2.1 Qubit and quantum state

A **qubit** is a unit of information describing a two-dimensional quantum system. It is the quantum counterpart of a bit for a classical system. A classical bit can either be in state 0 or 1, while a qubit can be in a state of **superposition** of the two bases $|0\rangle$ and $|1\rangle$. According to the Dirac notation, the states $|0\rangle$ and $|1\rangle$ are

defined as:

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}. \tag{1.1}$$

These two states form an orthogonal basis in $\mathbb{C}^2$, thus the set of all their linear combinations generates the vector space $\mathbb{C}^2$. The state $|\psi\rangle$ of a generic qubit, can be expressed as a linear combination of the basis states:

$$|\psi\rangle = c_0 |0\rangle + c_1 |1\rangle . \tag{1.2}$$

Therefore, $|\psi\rangle$ can be considered as simultaneously in both of these states (superposition), but, if a projective measurement onto the basis $\{|0\rangle , |1\rangle\}$ is performed, it will collapse into either $|0\rangle$ or $|1\rangle$ with probability $P(0) = |\langle 0|\psi\rangle|^2 = |c_0|^2$ and $P(1) = |\langle 1|\psi\rangle|^2 = |c_1|^2$, which are the projections of $|\psi\rangle$ onto $|0\rangle$ and $|1\rangle$ respectively. By definition of probability, $P(0) + P(1) = 1$, hence the state vector always has to be normalized, that is applying the following operation:

$$|\psi'\rangle = \frac{|\psi\rangle}{|| |\psi\rangle ||^2} \tag{1.3}$$

The vector is always normalized and multiplying it by a constant number does not affect it, thus the vector obtained is just a representation of the same state.

The state of a qubit can be written by expressing complex numbers with polar coordinates. In order to do this, it is necessary to start writing the probability amplitude in phase and modulus form:

$$|\psi\rangle = r_0 e^{i\phi_0} |0\rangle + r_1 e^{i\phi_1} |1\rangle . \tag{1.4}$$

Since a multiplication for a constant complex number gives just a different representation of the same normalized state, the amplitude of the first basis state can be made real by multiplying the state by a complex number with modulus 1 and phase $-\phi_0$:

$$e^{-i\phi_0} |\psi\rangle = e^{-i\phi_0}(r_0 e^{i\phi_0} |0\rangle + r_1 e^{i\phi_1} |1\rangle) = r_0 |0\rangle + r_1 e^{i(\phi_1 - \phi_0)} |1\rangle . \tag{1.5}$$

Furthermore, knowing that $|c_0|^2 + |c_1|^2 = r_0^2 + r_1^2 = 1$, the real parts of the amplitudes can be expressed as $r_0 = \cos\theta$ and $r_1 = \sin\theta$. Therefore, the polar form of the state is:

$$|\psi\rangle = \cos\frac{\theta}{2} |0\rangle + e^{i\phi} \sin\frac{\theta}{2} |1\rangle . \tag{1.6}$$

By exploiting the polar form, it is possible to graphically represent the state vector in the **Bloch sphere** (Figure 1.1). It is a sphere of radius 1, in which each point on its surface is an eligible quantum state. Any state $|\psi\rangle$ can be drawn on the sphere by means of the coordinates given by the Bloch vector($\mathbf{r}$):

$$\mathbf{r} = \begin{bmatrix} \sin\theta\cos\phi \\ \sin\theta\sin\phi \\ \cos\theta \end{bmatrix}. \tag{1.7}$$

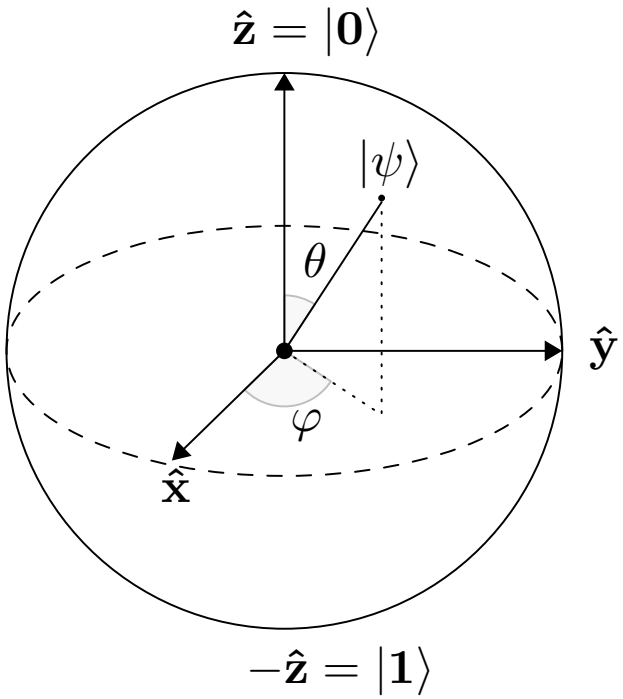


Figure 1.1: Bloch Sphere

The two basis states $|0\rangle$ and $|1\rangle$ are associated, respectively, with the north pole and the south pole. Any other point on the surface of the sphere is associated with a state composed of a superposition of the basis states. In particular, in the equator, all the states are a uniform superposition of $|0\rangle$ and $|1\rangle$ and they differ from each other just for the local phase ϕ.

A new quantum system can be obtained by assembling many quantum systems and states will be represented in a vector space that is the tensor product of the vector space of each system. Therefore, to obtain a system composed of more qubits, the quantum state can be described by the tensor product of the states of every single qubit. For example, a qubyte describes a system made by 8 qubits, whose vector space is isomorphic to $(\mathbb{C}^2)^{\otimes 8}$. The basis states of an assembled quantum system are the tensor product of the basis states of the systems that compose it. Instead, a state vector of an assembled system can not be always written as the tensor product of the states of the original systems. Take for example a state of the system obtained by assembling two systems A and B (called **Bell state**):

$$|\psi_{AB}\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle). \tag{1.8}$$

Writing this state as the tensor product of the states of A and B:

$$\begin{aligned} |\psi_{AB}\rangle =& |\phi_A\rangle \otimes |\phi_B\rangle = (c_0 |0\rangle + c_1 |1\rangle) \otimes (c_0' |0\rangle + c_1' |1\rangle) \\ =& c_0 c_0' |00\rangle + c_0 c_1' |01\rangle + c_1 c_0' |10\rangle + c_1 c_1' |11\rangle . \end{aligned} \tag{1.9}$$

Now to obtain the above state of the system AB, the amplitudes of the states A and B should satisfy the equations $c_0c'_0 = c_1c'_1 = 1$ and $c_0c'_1 = c_1c'_0 = 0$, but they have no solution. Therefor,e the state of an assembled system can not always be written as a tensor product. States for which this is possible are called **separable states**, whereas states for which this is not, are called **entangled states**. The Bell state previously shown is obtained creating entanglement between qubits. For the state $|\psi_{AB}\rangle$, if a measurement finds the first qubit in the state $|0\rangle$ the second qubit collapses to the state $|0\rangle$ since there is a probability equal to 1 to measure the second qubit in the state $|0\rangle$. Equivalently if a measurement finds the first qubit in the state $|1\rangle$ the state of the second qubit collapses to $|1\rangle$. **Entanglement** is a very strong form of correlation because measuring one qubit, even if the other system is in a faraway position, it is possible to know the state of the other qubit.

1.2.2 Quantum gates

A **quantum circuit** is described by a sequence of gates, that perform elementary operations on qubits. Unlike classical gates, quantum ones are not physical blocks inserted and connected to other blocks in the circuit, but they are electric or magnetic pulses applied to qubits characterized by amplitude, phase and time of application [9] and allow performing any reversible transformation. Reversibility means that inputs can be derived from outputs. Consequently, there is a one-to-one correspondence between inputs and outputs. Since the operators applied to quantum systems are characterized by this property, quantum gates are represented by unitary matrices (for a unitary matrix U: $UU^\dagger = U^\dagger U = I$, where I is the identity matrix).

The fundamental single qubits gates are the **Pauli gates**. These gates, visualizing their effect on the Bloch sphere, perform a rotation of an angle π around the x, y and z axis, respectively. For the states $|0\rangle$ and $|1\rangle$ applying the Pauli X gate is equivalent to the classical not operation because rotating the north pole vector by π, the south pole vector is obtained and vice versa. Indeed, the Pauli X gate is also called NOT. X, Y and Z transformations do not affect states parallel respectively to the x, y and z axes in the Bloch sphere. The corresponding matrices are reported below:

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \; Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}, \; Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}. \tag{1.10}$$

The S **gate** performs a rotation of $\pi/2$ around the z-axis, thus the corresponding matrix is the square root of the Pauli Z gate:

$$S = \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}. \tag{1.11}$$

The **Hadamard gate** can create a superposition of states if applied to a basis state $|0\rangle$ or $|1\rangle$, indeed it is often used at the beginning of quantum algorithms for the

state preparation. It can be interpreted as a rotation of $\pi/2$ around the y-axis and then a rotation of π around the x-axis, therefore $H = XY^{\frac{1}{2}}$.

$$H = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \tag{1.12}$$

$$H\,|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)\,. \tag{1.13}$$

The gates just examined are all inversions or rotations of a fixed angle on the Bloch sphere, but **phase shift gates** allow for turning the state vector on the surface of the Bloch sphere around each direction of any angle θ, that is a parameter of the gate.

$$R_x(\theta) = \begin{bmatrix} \cos\frac{\theta}{2} & -i\sin\frac{\theta}{2} \\ -i\sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{bmatrix}, \tag{1.14}$$

$$R_y(\theta) = \begin{bmatrix} \cos\frac{\theta}{2} & -\sin\frac{\theta}{2} \\ \sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{bmatrix}, \tag{1.15}$$

$$R_z(\theta) = \begin{bmatrix} e^{-i\frac{\theta}{2}} & 0 \\ 0 & e^{i\frac{\theta}{2}} \end{bmatrix}. \tag{1.16}$$

A gate acting on $n + m$ qubits can be used to perform an operation on n qubits (targets) according to the state of the remaining m qubits (controls). A gate that does such an operation is called **controlled gate**. The identity operation is performed on the target qubits, If control is applied to a qubits in state $|0\rangle$. Otherwise, the quantum gate is enforced on the targets. Alternatively, if anti-control is applied to a qubit the quantum gate is enforced to the other n ones, if it is in state $|0\rangle$, or, in the opposite case, the identity operation is performed.

The controlled-U (or ${}^cU$) operation can be formalized for any quantum gate U and it is represented by the following unitary matrix considering the control to be applied to the most significant qubit:

$$U = \begin{bmatrix} a & b \\ c & d \end{bmatrix},\ {}^cU = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & a & b \\ 0 & 0 & c & d \end{bmatrix}. \tag{1.17}$$

For instance, the controlled not gate (or CNOT) acting on 2 qubits when the control is applied to the most significant qubit is represented by the following matrix:

$$CNOT = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}. \tag{1.18}$$

In quantum computing, the circuit can be just a visual representation of the sequence of gates to be applied to the qubits. Gates can just be stimuli applied to the qubits and they are not physical blocks connecting them. Hence, the quantum equivalent of swapping two wires is the **swap gate** [10]. It can be used for addressing issues of connectivity among qubits. Since multiple qubits operations, in general, can not be performed on any group of qubits, the swap can be utilized to transfer the state of a qubit to different physical qubits.

$$SWAP = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{1.19}$$

A swap gate does not affect input states $|00\rangle$ and $|11\rangle$, whereas it complements the inputs $|01\rangle$ and $|10\rangle$. It can be constructed only using CNOT gates (Figure 1.2). In the figure on the left-hand side, it is shown the symbol of the swap gate, whereas, on the right-hand side, the three CNOT gates required for implementing it are represented by the corresponding symbols. If the input is $|01\rangle$ the last gate will not have any effect and both qubits will be complemented, if it is $|10\rangle$ the first gate will not have an effect and both qubits will be complemented and if the inputs are $|00\rangle$ or $|11\rangle$ the gates will not have an effect at all since states are symmetric.

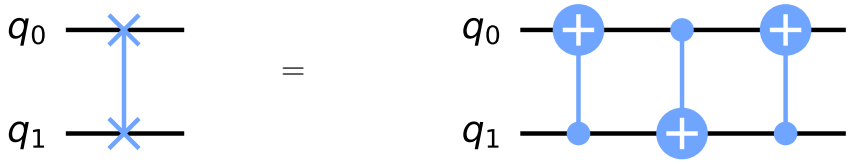


Figure 1.2: Equivalence between swap and three CNOT gates

Finally, the measurement operation is employed at the end of a circuit to measure qubits and obtain classical bits. It is indicated by the symbol in Figure 1.3.

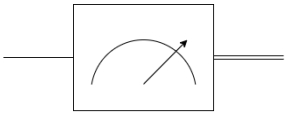


Figure 1.3: Measurement symbol

1.2.3 Quantum algorithms

A quantum algorithm is a finite sequence of steps to solve a given problem, running on a quantum computer, thus exploiting quantum mechanics properties to achieve a lower computational complexity over any other classical algorithm. A typical element found in many quantum algorithms is an oracle, which can provide some answers by passing it queries. For instance, an oracle could give a boolean answer to the question "is an input configuration x present in a certain database?". More precisely, an **Oracle** is a "black box" operation that evaluates a function but it is not possible to see how this function is defined. In a classical computer, an oracle could be defined as an operator, that can be written as a matrix acting on the input bits, which has an input string on n bits and an output string on m bits:

$$f : \{0,1\}^n \to \{0,1\}^m. \tag{1.20}$$

On a quantum computer, each operator should be unitary and reversible, therefore if the function has n qubits as input and m qubits as output the quantum gate needs another $m-n$ qubits as input called **ancillae qubits**, since from the output it must be possible to derive the input. Therefore, a **boolean oracle** is defined as:

$$O_f(|x\rangle \otimes |y\rangle) = |x\rangle \otimes (|y\rangle \oplus f(x)). \tag{1.21}$$

Using the boolean oracle, initializing the ancilla qubit to $|1\rangle$ and then applying a Hadamard gate, a **phase oracle** can be constructed. According to the value of the function, it can apply a phase to the input:

$$U_f \, |x\rangle = (-1)^{f(x)} \, |x\rangle \, . \tag{1.22}$$

In this context, the ancilla qubit is needed to implement the function but the result is independent of it. Hence, it is not reported and it is considered included in the phase oracle operator.

1.3 Deutsch-Josza algorithm

The first quantum algorithm that, by using the superposition of states, showed an advantage with respect to the classical ones, is the **Deutsch-Josza algorithm**. Given an oracle that evaluates a function $f : \{0,1\}^n \to \{0,1\}$ which can be either balanced or constant, the Deutsch-Josza algorithm is able to determine which of the two properties f possesses using just one query. For classical computing in the worst case, $2^{n-1} + 1$ queries are needed, where n is the number of input bits. The algorithm implemented by quantum gates is reported in Figure 1.4.

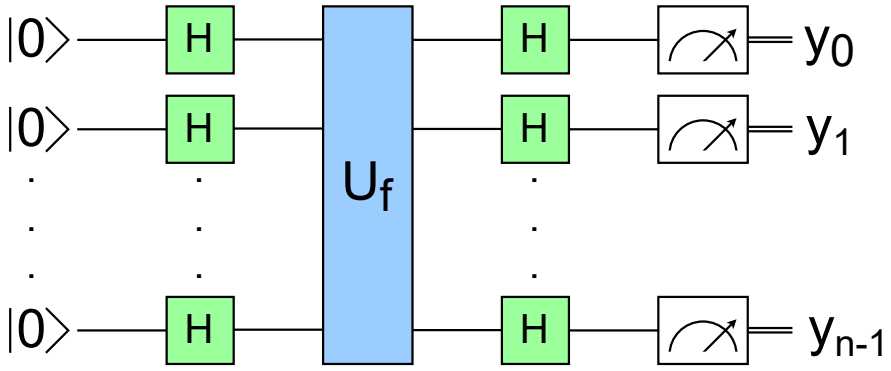


Figure 1.4: Deutsch-Josza algorithm

If the function is constant, if $f(x) = 0$, the phase oracle does not affect the input states. Two Hadamard gates applied in series, since they are unitary matrices, are equivalent to applying an identity matrix. If $f(x) = 1$, the oracle applies a global phase of -1, but, for the measurements, the global phase has no meaning, so the input states are again not affected. Instead, if the function is balanced, the phase oracle actually affects the inputs and the outputs will be different than the 0 string.

1.4 Grover's algorithm

Grover's search has been the first quantum algorithm to prove the quantum advantage. It searches one or more elements in an unsorted database composed of N elements with a computational cost that is $O(\sqrt{N})$, whereas on average the classical counterpart has $O(N)$ complexity. In the beginning, the **state preparation** operation creates a superposition of all the states in the database. It can be made up of only Hadamard gates to generate an equal superposition of all the possible states. A phase oracle U_f implements a function that returns 1 if the input state is equal to the winning element $|w\rangle$, which is the element having the desired characteristics. This oracle changes the global phase of the state only if it is equal to $|w\rangle$. Then another phase oracle U_{f_0} is needed that returns 0 if the input state is the 0-bit string, so it changes the global phase of the state if it is different from the 0 string.

The implementation of Grover's algorithm is reported in Figure 1.5 and the block formed by the inverse state preparation, in this case Hadamard gates, at the inputs, U_{f_0} and the state preparation at the outputs, is called **Diffuser**($\mathbf{V}$).

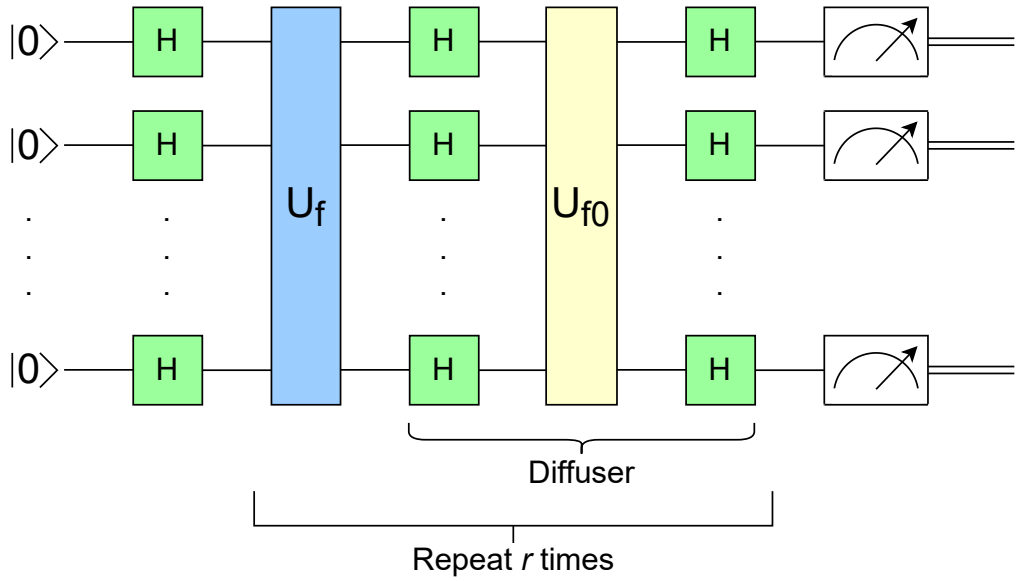


Figure 1.5: Grover's algorithm

The first step in the algorithm is the state preparation. It encodes the database in the quantum state in which the algorithm has to search for the winning element. The simplest state preparation operator is composed of a layer of Hadamard gates that prepares an initial state $|s\rangle$ composed of the equal superposition of all the possible input configurations. Therefore, each state has the same amplitude, as shown in Figure 1.6.

$$|s\rangle = \frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} |x\rangle \,. \tag{1.23}$$

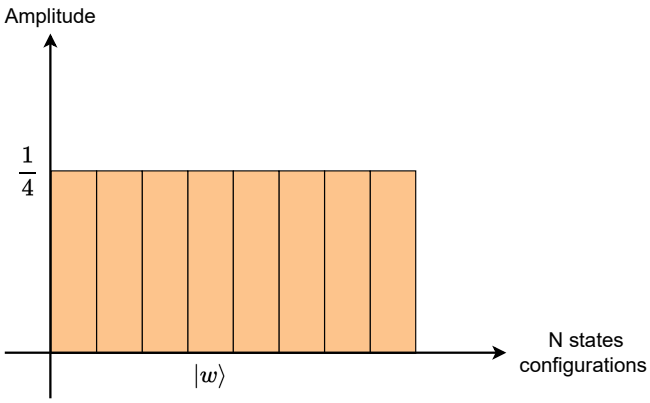


Figure 1.6: States amplitude after $H^{\otimes n}$ operator (example with 3 qubits)

The functions evaluated by the two oracles can be written as shown in the

following.

$$U_f = \mathbb{I} - 2\,|w\rangle\langle w|\,. \tag{1.24}$$

The phase oracle U_f flips the global phase only of the winning element because the projection of a state $|x\rangle$ on $|w\rangle$ is 0, since they are orthogonal, whereas it gives $\mathbb{I}\,|w\rangle - 2\,|w\rangle = -\,|w\rangle$ for the target entry $|w\rangle$. This operation is called **phase inversion**: the amplitude of the element with the desired characteristics is flipped and becomes negative (Figure 1.7).

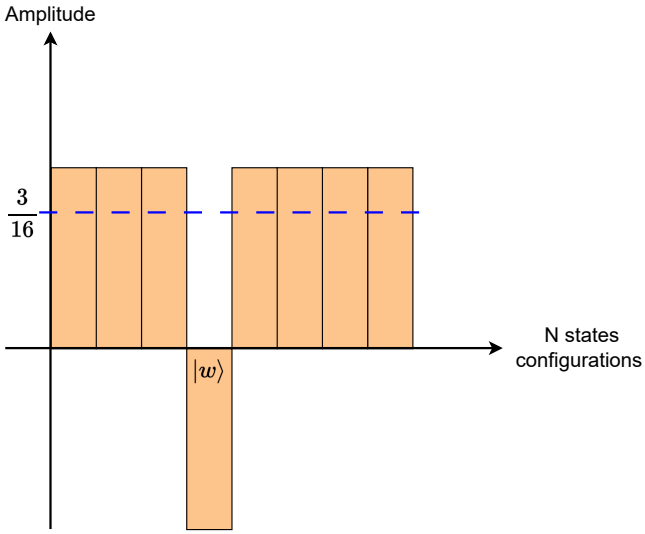


Figure 1.7: States amplitude after U_f operator (example with 3 qubits)

$$U_{f_0} = 2\,|0\rangle\langle 0|^{\otimes n} - \mathbb{I}. \tag{1.25}$$

The state preparation composed of only Hadamard gates creates the equal superposition of all the possible states:

$$|s\rangle = \sum_{x=0}^{2^n} \frac{1}{\sqrt{2^n}}\,|x\rangle\,. \tag{1.26}$$

The phase oracle U_{f_0} flips the global phase of all the states except the 0 string because only for the $|0\rangle^{\otimes n}$ state the projection onto itself is different than 0. The diffuser function can now be evaluated by substituting the U_{f_0} expression.

$$V = H^{\otimes n} U_{f_0} H^{\otimes n} = 2H^{\otimes n}\,|0\rangle\langle 0|^{\otimes n}\,H^{\otimes n} - H^{\otimes n}\mathbb{I}H^{\otimes n} = 2\,|s\rangle\langle s| - \mathbb{I}. \tag{1.27}$$

Indicating as $|w^{\perp}\rangle$ the superposition of every state except $|w\rangle$, that is a state which lies in the plane spanned by $|s\rangle$ and $|w\rangle$, but is orthogonal to $|w\rangle$, the state $|s\rangle$ can

be written in the following way:

$$|s\rangle = \sqrt{\frac{2^n - 1}{2^n}} |w^\perp\rangle + \frac{1}{\sqrt{2^n}} |w\rangle = \cos\frac{\theta}{2} |w^\perp\rangle + \sin\frac{\theta}{2} |w\rangle . \tag{1.28}$$

Therefore, applying U_f means applying the identity to $|s\rangle$ and then subtracting two times the projection of $|s\rangle$ into $|w\rangle$, so this is equivalent to a reflection of $|s\rangle$ around $|w^\perp\rangle$. Then applying the diffuser is equivalent to a reflection of the state obtained before around $|s\rangle$, so the VU_f operation corresponds to a rotation of $|s\rangle$ by an angle θ [11, Sec. 3.8] as shown in the Figure 1.8.

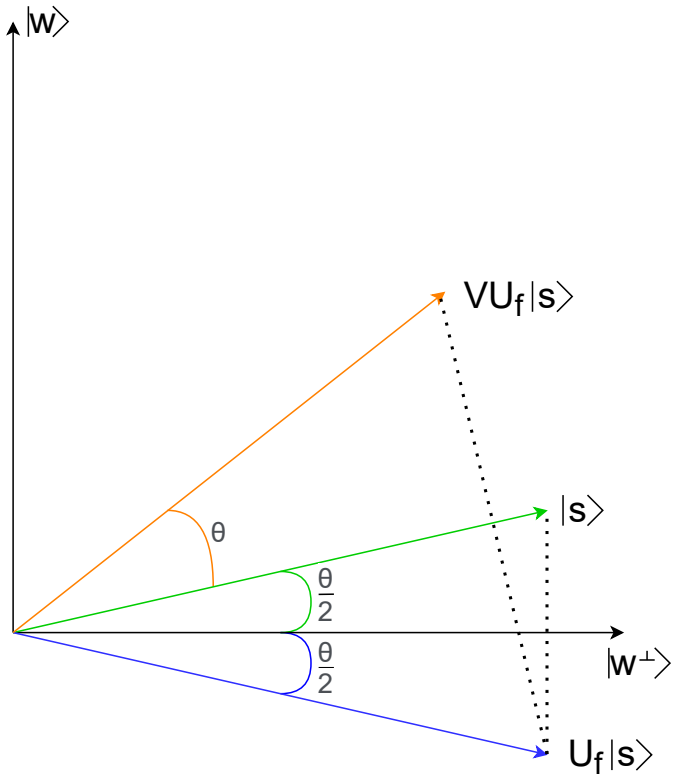


Figure 1.8: Effect of phase oracle and diffuser in Grover's algorithm

Observing the diffusion operation by the amplitudes of all possible states' point of view, it is performed an inversion about the average. Since $|s\rangle$ is the equal superposition of all the basis states, each of them has the same amplitude $\frac{1}{\sqrt{2^n}}$, therefore the outer product $|s\rangle\langle s|$ is a matrix A that for each element has $(\frac{1}{\sqrt{2^n}})^2 = \frac{1}{2^n}$. Hence, it is the matrix that finds the average ($\langle\alpha\rangle$) of the amplitudes of the state ($A|\psi\rangle = \langle\alpha\rangle$). Applying the diffuser on a state $|\psi\rangle = \sum_{x\in\{0,1\}^n} \alpha_x |x\rangle$ means

applying the operator $2A - \mathbb{I}$, which gives as a result:

$$V|\psi\rangle = \sum_{x\in\{0,1\}^n} (-\alpha_x + 2\langle\alpha\rangle)|x\rangle = \sum_{x\in\{0,1\}^n} (\langle\alpha\rangle + (\langle\alpha\rangle - \alpha_x))|x\rangle \tag{1.29}$$

The result obtained is, indeed, the inversion about the average of the amplitudes of all the possible states [12] (Figure 1.9).

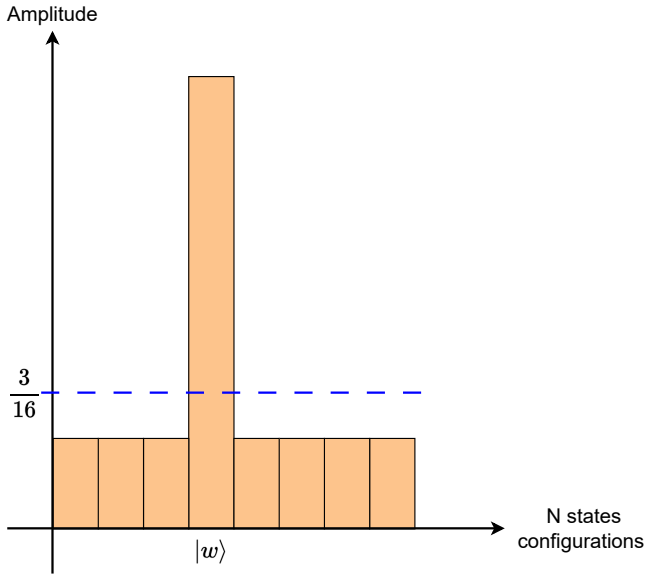


Figure 1.9: States amplitude after V operator (example with 3 qubits)

The whole operation performed by the phase oracle and the diffuser is called **amplitude amplification**. In this way, the amplitude of all the possible states is reduced, whereas the amplitude of the winning element is increased. Iterating this process means increasing the amplitude of the winning element, thus increasing the probability of measuring it.

Once the phase oracle and diffusion operations are repeated r times, the initial state $|s\rangle$ has been rotated by an angle $r\theta$. The goal is to rotate the initial state in order to get as close as possible to the state $|w\rangle$ from a starting angle of $\frac{\theta}{2}$, thus $r\theta = \frac{\pi}{2} - \frac{\theta}{2}$. For the expression of the initial state $|s\rangle$: $\sin\frac{\theta}{2} = \frac{1}{\sqrt{2^n}}$. Hence it is possible to evaluate the optimal r, knowing θ:

$$r = \frac{\pi}{2\theta} - \frac{1}{2} = \frac{\pi}{4\arcsin\frac{1}{\sqrt{2^n}}} - \frac{1}{2}. \tag{1.30}$$

If the number of qubits n and, as a consequence, the number of elements $N = 2^n$, is sufficiently high to approximate r:

$$r \approx \frac{\pi}{4}\sqrt{2^n}. \tag{1.31}$$

This shows that the number of iterations of the Grover search algorithm scales as $O(\sqrt{N})$. After r iterations, the final state $|y\rangle$ and the state $|w\rangle$ are at most separated by an angular difference of $\frac{\theta}{2}$ because otherwise another rotation would have been performed to get closer to $|w\rangle$. Therefore, the probability to measure the winning element is given by:

$$P(w) = |\langle w|y\rangle|^2 \geq \cos^2\frac{\theta}{2} = 1 - \frac{1}{2^n}. \tag{1.32}$$

Grover search algorithm can be also used to mark multiple target elements, obtaining as the final state the superposition of all the winning strings [13]. Calling M the number of elements to mark:

$$\begin{aligned} |w\rangle &= \frac{1}{\sqrt{M}}\sum_{i=1}^{M}|w_i\rangle\,, \\ |w^\perp\rangle &= \frac{1}{\sqrt{2^n - M}}\sum_{x\notin\{w_1,\ldots,w_M\}}|x\rangle\,, \\ |s\rangle &= \sqrt{\frac{2^n - M}{2^n}}\,|w^\perp\rangle + \sqrt{\frac{M}{N}}\,|w\rangle = \cos\frac{\theta}{2}\,|w^\perp\rangle + \sin\frac{\theta}{2}\,|w\rangle\,. \end{aligned} \tag{1.33}$$

The initial state $|s\rangle$ can be expressed in the same way as before, but the value of θ has now changed. Grover's iterations still perform a rotation of an angle θ towards the winning state, thus, also in this case, r iterations are needed to rotate the state by an angle $\frac{\pi}{2} - \frac{\theta}{2}$.

$$r = \frac{\pi}{4\arcsin\sqrt{\frac{M}{2^n}}} - \frac{1}{2}. \tag{1.34}$$

Considering a very high number of elements $N = 2^n$ with respect to the number of winning ones, the optimal number of rotations can be approximated as:

$$r \approx \frac{\pi}{4}\sqrt{\frac{2^n}{M}}. \tag{1.35}$$

Note that a particular case is when the number of solutions is exactly half of the database size. In this case, the state $|s\rangle$ is composed of an equal superposition of winning and non-winning elements, therefore independently of the number of iterations $\theta = \pi/4$. This implies an equal probability of finding any item among all the input configurations.

1.5 Quantum Fourier Transform

The **Quantum Fourier Transform** (**QFT**) [11, Sec. 3.5] performs the quantum counterpart of the Discrete Fourier Transform. Considering $N = 2^n$, where n is the

number of qubits:

$$QFT_N \left|\psi\right\rangle = \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} \exp\left\{2\pi i \frac{\psi x}{N}\right\} \left|x\right\rangle . \tag{1.36}$$

This function transforms qubits from the Z-basis (or computational) to the Fourier basis (Figures 1.10 and 1.11). Therefore, it takes as inputs binary numbers and encodes them in rotations across the z-axis, so that the most significant qubit is rotated by $\frac{2\pi\psi}{N}$, the qubit after by the double of the angle ($\frac{4\pi\psi}{N}$), the one after that by the double of the latter angle and so on.

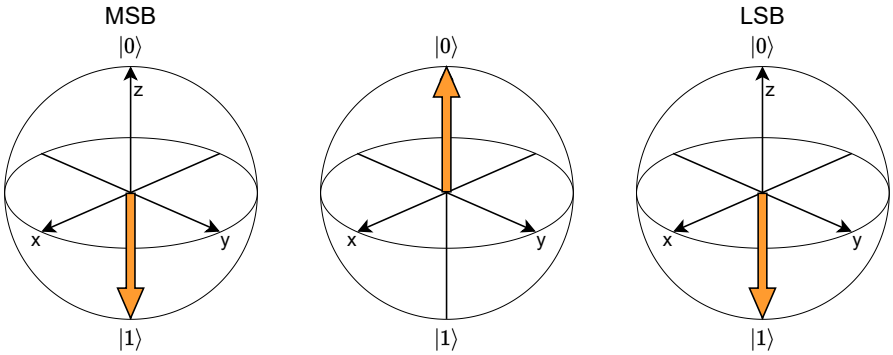


Figure 1.10: Number $-3 = (101)_2$ encoded in the z-basis in 2's complement

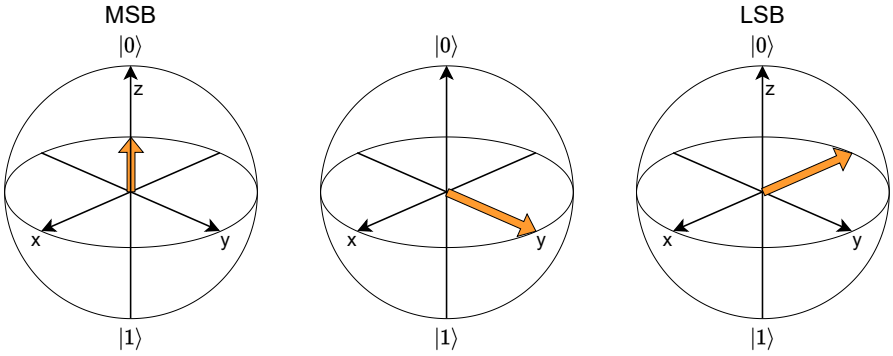


Figure 1.11: Number $-3 = (101)_2$ encoded in the Fourier basis. The most significant qubit, on the left, is rotated by $\frac{5\times 2\pi}{8} = \frac{5}{4}\pi$. The middle qubit is rotated by $\frac{10\times 2\pi}{8} = \frac{5}{2}\pi = \frac{\pi}{2}$. The least significant qubit is rotated by $5\pi = \pi$.

The QFT circuit exploits the controlled phase gate ${}^{C}P_k$ gate, defined as follows:

$${}^{C}P_k = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & \exp\left\{\frac{2\pi i}{2^k}\right\} \end{bmatrix} . \tag{1.37}$$

To the first qubit in the circuit (Figure 1.12), that is the least significant one, a Hadamard gate is applied, so the state becomes:

$$H_1 |\psi_0\rangle = \frac{1}{\sqrt{2}} \left[|0\rangle + \exp\left\{ \frac{2\pi i}{2} \psi_0 \right\} |1\rangle \right]. \tag{1.38}$$

Then $n-1$ ${}^{C}P_i$ gates are applied by using as control qubits the $n-1$ more significant ones. The controlled phase gate applies the phase change only to the state $|1\rangle$ if the control qubit is 1. After that, the state becomes:

$$\begin{aligned} &\frac{1}{\sqrt{2}} \left[|0\rangle + \left(\exp\left\{ \frac{2\pi i}{2^n} \psi_{n-1} \right\} + ... + \exp\left\{ \frac{2\pi i}{2} \psi_0 \right\} \right) |1\rangle \right] \otimes |\psi_1 ... \psi_{n-1}\rangle = \\ &\frac{1}{\sqrt{2}} \left[|0\rangle + \exp\left\{ \frac{2\pi i}{2^n} \psi \right\} |1\rangle \right] \otimes |\psi_1 ... \psi_{n-1}\rangle . \end{aligned} \tag{1.39}$$

Successively the same gates are applied to any other qubit. Therefore, to each qubit, following the order from the least significant to the most significant, it is applied a Hadamard gate first and then phase gates controlled by all the more significant qubits than the one at hand. The obtained state is:

$$\begin{aligned} &\frac{1}{\sqrt{2}} \left[|0\rangle + \exp\left\{ \frac{2\pi i}{2^n} \psi \right\} |1\rangle \right] \otimes \frac{1}{\sqrt{2}} \left[|0\rangle + \exp\left\{ \frac{2\pi i}{2^{n-1}} \psi \right\} |1\rangle \right] \otimes ... \\ &\otimes \frac{1}{\sqrt{2}} \left[|0\rangle + \exp\left\{ \frac{2\pi i}{2} \psi \right\} |1\rangle \right]. \end{aligned} \tag{1.40}$$

In the end, a layer of $n/2$ swap gates is needed to reverse the order of the qubits. The first is swapped with the last, the second with the second to last and so on. Now expressing this result in terms of the 2^n basis states the same expression of the discrete Fourier transform is attained:

$$\frac{1}{\sqrt{2^n}} \bigotimes_{k=1}^{n} \left[|0\rangle + \exp\left\{ \frac{2\pi i \psi}{2^k} \right\} |1\rangle \right] = \frac{1}{\sqrt{2^n}} \sum_{x=0}^{N-1} \exp\left\{ 2\pi i \frac{\psi x}{2^n} \right\} |x\rangle . \tag{1.41}$$

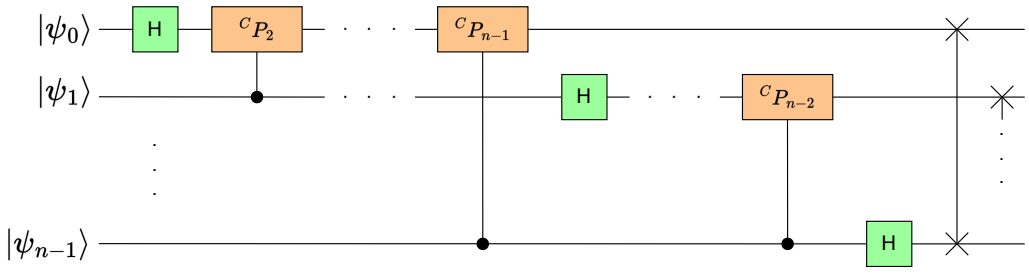


Figure 1.12: Circuit that implements QFT algorithm

Chapter 2

QUBO formulation

2.1 Formulation

Quadratic unconstrained binary optimization (**QUBO**) is a mathematical formulation, which can be exploited for describing many real-world combinatorial problems. It is the most feasible formulation for solving optimization problems employing quantum approaches. In the acronym, quadratic indicates that the highest power appearing in the function is two, unconstrained means that no constraint is formally applied to variables, binary implies that variables can only assume values 0 and 1, and optimization refers to the target of the description, i.e. maximizing or minimizing an objective function. Therefore, this model can be expressed in the following way:

$$\text{minimize } f(\mathbf{x}) = q_0 + \sum_i q_i x_i + \sum_{i<j} q_{ij} x_i x_j, \tag{2.1}$$

where q_0 is a constant term and q_i and q_{ij} are the coefficients of linear and quadratic terms, respectively. The model can also be written as:

$$\text{minimize } f(\mathbf{x}) = \mathbf{x}^t Q \mathbf{x}, \tag{2.2}$$

where $\mathbf{x}$ is a vector of binary variables and Q is a square matrix of problem coefficients. The Q matrix can be **symmetric** or in **upper triangular form**. If it is in upper triangular form for all i and j with $j > i$, the element in the row i and column j of the matrix is equal to the coefficient q_{ij}. If it is symmetric, for all i and j the element in the row i and column j is replaced by $q_{ij}/2$. In both cases, linear terms are located in the diagonal.

$$Q_{triangular} = \begin{bmatrix} q_0 & q_{01} & q_{02} & q_{03} \\ 0 & q_1 & q_{12} & q_{13} \\ 0 & 0 & q_2 & q_{23} \\ 0 & 0 & 0 & q_3 \end{bmatrix}, \; Q_{symmetric} = \begin{bmatrix} q_0 & \frac{q_{01}}{2} & \frac{q_{02}}{2} & \frac{q_{03}}{2} \\ \frac{q_{01}}{2} & q_1 & \frac{q_{12}}{2} & \frac{q_{13}}{2} \\ \frac{q_{02}}{2} & \frac{q_{12}}{2} & q_2 & \frac{q_{23}}{2} \\ \frac{q_{03}}{2} & \frac{q_{13}}{2} & \frac{q_{23}}{2} & q_3 \end{bmatrix}. \tag{2.3}$$

2.2 Constraints

In many optimization problems, the admissible solutions can be less than the possible ones because they may have to satisfy some conditions, that are called constraints. The QUBO model formally does not explicitly consider constraints. However, they can be taken into account by introducing weighted **quadratic penalties** in the QUBO cost function. Therefore, extra components $g(x)$ multiplied by a positive coefficient P are added to the original function:

$$y = f(x) + Pg(x)\,. \tag{2.4}$$

Penalties are chosen such that the cost function gives higher results for solutions that do not satisfy constraints. Hence, the penalty functions value is equal to 0 for feasible solutions, whereas it is equal to a positive amount for unfeasible ones [14]. For some very common constraints, quadratic penalty functions are known in advance and reported in Table 2.1.

Classical constraint	Equivalent penalty
$x + y \leq 1$	$P(xy)$
$x + y \geq 1$	$P(1 - x - y + xy)$
$x + y = 1$	$P(1 - x - y + 2xy)$
$x \leq y$	$P(x - xy)$
$x_1 + x_2 + x_3 \leq 1$	$P(x_1x_2 + x_1x_3 + x_2x_3)$
$x = y$	$P(x + y - 2xy)$

Table 2.1: Quadratic penalty functions of common employed constraints [14]

For general cases, in which penalty functions are not known in advance, it is possible to find them by following the procedure reported below. Given a general binary optimization problem:

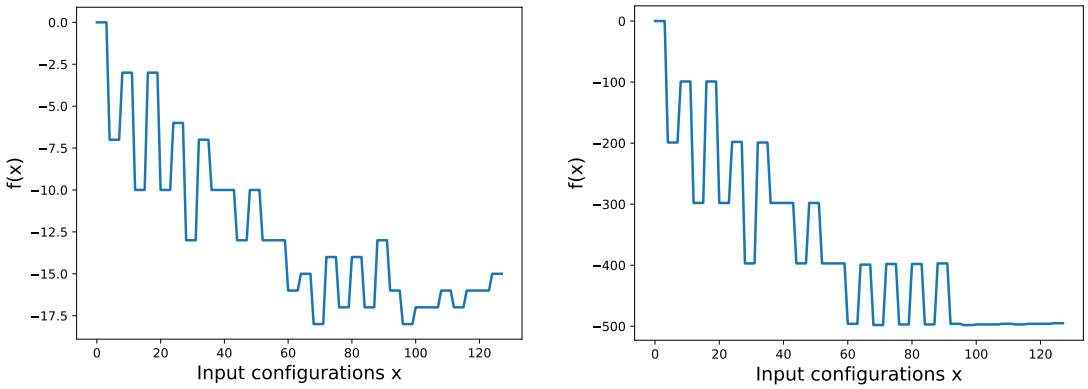
$$\begin{aligned} &\min f(x) = x^t C x, \\ &\text{subject to: } Ax = b. \end{aligned} \tag{2.5}$$

Constraints expressed as a linear system can be always written as quadratic penalties to be added to the cost function. At this point a new QUBO model, taking into account constraints, can be derived:

$$f(x) = x^t C x + P(Ax - b)^t(Ax - b) = x^t C x + x^t D x + c = x^t Q x + c. \tag{2.6}$$

Dropping the additive constant, the now unconstrained problem is in the original QUBO form. The constant c does not affect the optimal solution and, consequently, can be neglected in the optimization phase. It can be eventually added once the solution is found to recover the original cost function value.

Penalty coefficient values are crucial since a too-high value can make the function profile flatter thus making it more difficult to distinguish a better solution from a worse one. Instead, a too-small one may cause a configuration of variables that violates some constraints to have a better or equal result than other feasible solutions [14]. Figure 2.1 displays an example of the profile of two functions obtained with low and high P values.



(a) $P = 4$ (b) $P = 100$

Figure 2.1: Comparison of a cost function profile for a minimum vertex cover problem (described in section 2.4) with (a) low P value and (b) high P value

Constraints can be of two categories: hard and soft. For a hard one, violations are not admitted, thus P must be sufficiently large. For a soft one, violations are accepted, hence the coefficient P can be smaller. Consider that the methodology provided for including constraints in the QUBO problem is fully general because, if A and b have integer elements, inequality constraints can always be traced back to $Ax = b$ form using **slack variables**. For example, the constraint $4x_1 + 5x_2 - x_3 \leq 6$ can be transformed into an equality constraint by adding a variable s, that can be represented through a binary expansion, leading to $4x_1 + 5x_2 - x_3 + s = 6$. Then, the number of binary variables required for the expansion depends on the upper bound reachable by s. In the example, since $s \leq 7$, a possibility is to expand it with three binary variables (logarithmic expansion): $s_1 + 2s_2 + 4s_3$. Then, the slack variables have to be included in the binary variables vector x and the slack coefficients in the matrix A.

2.3 QUBO and Ising models equivalence

The **Ising model** [15] was initially conceived for describing the magnetic properties of some materials. The variables represent the spin of atoms that can be in spin-up or spin-down configuration, which correspond to the values $+1$ and -1, respectively.

Atoms are located in a lattice and can interact in two different ways:

- external field interaction (linear) which determines the orientation preferred by a single spin;
- quadratic interaction between a couple of spins, which indicates if they prefer an aligned or anti-aligned orientation.

The system Hamiltonian is the following:

$$H(s) = \sum_i h_i s_i + \sum_{i<j} J_{ij} s_i s_j, \tag{2.7}$$

where h_i are the external field coefficients and J_{ij} are the interaction coefficients. The variables s represent the spin of atoms. It is possible to notice that this model can be employed for expressing combinatorial optimization problems. It is perfectly equivalent to the QUBO formulation, indeed, the only difference is that the binary variables for the Ising one are bipolar instead of unipolar (0, 1). In order to convert the QUBO model into an Ising one, the following relation can be exploited:

$$s_i = 2x_i - 1, \tag{2.8}$$

whereas to transform the Ising model into a QUBO one, the equation is reported below:

$$x_i = \frac{1 + s_i}{2}. \tag{2.9}$$

Moreover, solving QUBO problems is very important since any pseudo-boolean function, which is a function $f : \mathbb{B}^n \to \mathbb{R}$, can be represented as a multilinear polynomial in its variables, and every multilinear polynomial binary optimization problem can be transformed in a quadratic one at the cost of adding auxiliary variables [16].

2.4 Benchmark problems

This section showcases the various problem classes, representable through the QUBO formulation, which have been employed for testing the algorithms presented in this thesis and obtaining the results presented in Chapter 8. The *qubovert* [17] Python library has been exploited to formulate all these problems and to compute the optimal value and input configurations using the available solvers.

2.4.1 Minimum vertex cover

Given an undirected graph $G(V, E)$, with vertex set V and edge set E, the **minimum vertex cover** is the smallest subgraph such that every edge in E has at

least one endpoint in one of the vertices of the subgraph [14] (example in Figure 2.2).

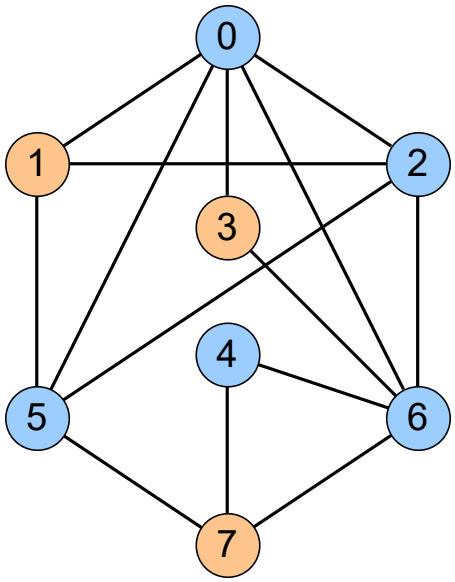


Figure 2.2: Example of minimum vertex cover (highlighted in blue).

The QUBO formulation can be written by associating a variable x_i with the i^{th} vertex which assumes the binary value 1 if the vertex belongs to the minimum vertex cover, or it is equal to 0 otherwise. It has to minimize the number of nodes in the subgraph, satisfying the condition that for each edge connecting nodes i and j, at least one of them must be part of the vertex cover. This can be written in the following way:

$$\begin{aligned} &\text{minimize } f(x) = \sum_{i \in V} x_i, \\ &\text{subject to: } x_i + x_j \geq 1 \ \forall (i,j) \in E. \end{aligned} \tag{2.10}$$

The constraint has the form of one of those shown in Table 2.1 and the corresponding quadratic penalty can be directly included in the objective function. Hence, the final QUBO function is obtained:

$$\text{minimize } f(x) = \sum_{i \in V} x_i + P \left(\sum_{(i,j) \in E} (1 - x_i - x_j + x_i x_j) \right) \tag{2.11}$$

For instance, a feasible value for the penalty coefficient is $P = 8$.

2.4.2 Maximum clique

Given an undirected graph $G(V, E)$, with vertex set V and edge set E, the **maximum clique** is the subgraph G' with the largest size, in which every vertex is

connected by an edge with all the other vertices in the subgraph (example in Figure 2.3).

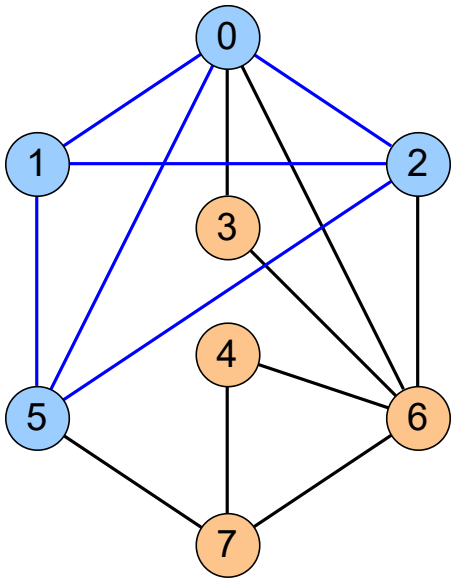


Figure 2.3: Example of maximum clique (highlighted in blue).

The QUBO formulation can be written by associating a variable x_i with the i^{th} vertex which assumes the binary value 1 if it is part of the clique, or it is equal to 0 otherwise. Note that this is a maximization problem, thus the optimal solution is the maximum value of the function, but then it will be considered as a minimization one by flipping the sign of the objective function once it is generated. The goal is to maximize the number of vertices in the subgraph, satisfying the condition that for each couple of vertices in G' an edge must exist. Therefore, the problem can be written in the following way:

$$\begin{aligned} &\text{minimize } f(x) = -\sum_{i \in V} x_i, \\ &\text{subject to: } x_i + x_j \leq 1 \ \forall (i,j) \in \overline{E}, \end{aligned} \tag{2.12}$$

where $\overline{E}$ indicates the set of couples of nodes not connected by an edge. Expressing the constraint as the quadratic penalty shown in Table 2.1 and including it in $f(x)$, the following QUBO formulation is obtained:

$$\text{minimize } f(x) = -\sum_{i \in V} x_i + P \left(\sum_{(i,j) \in \overline{E}} x_i x_j \right), \tag{2.13}$$

where the penalty coefficient is often chosen as $P = 2$.

2.4.3 Max-cut

Given an undirected graph $G(V, E)$, with vertex set V and edge set E, the **max-cut** target is to partition the graph in two subsets such that the sum of the edges that connects a vertex in one subset to a vertex in the other is the maximum one [14] (example in Figure 2.4).

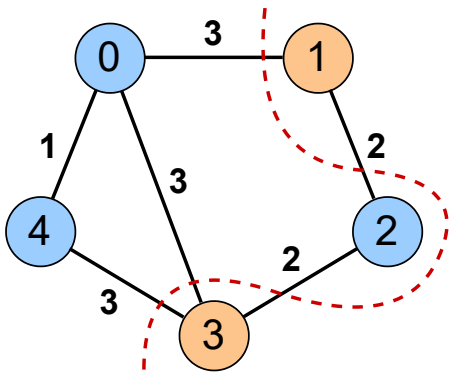


Figure 2.4: Example of maximum cut (subsets of vertices highlighted in orange and blue, cut shown by the red dashed line).

The QUBO formulation can be written by associating a variable x_i with the i^{th} vertex, which assumes the binary value 1 if it is part of one subset, or it is equal to 0 otherwise. Note that this is a maximization problem, thus it will be considered as a minimization one by flipping the sign of the objective function once it is generated. For each edge, it is desired that the objective function increases if the two vertices are in different subsets, thus there is a cut going through this edge, and it does not increase if the vertices are in the same subset. Therefore, the polynomial expression $x_i + x_j - 2x_i x_j$ is equal to 1 when the nodes, associated with those variables, are separated by a cut, whereas it is equal to 0 when they are in the same subset. By summing this expression for every edge, the following QUBO formulation is obtained:

$$\text{minimize } f(x) = \sum_{(i,j)\in E} (-x_i - x_j + 2x_i x_j) \tag{2.14}$$

Chapter 3

Grover Adaptive Search

The **Grover Adaptive Search** (GAS) is a hybrid quantum-classical successive approximation algorithm that looks for the optimal value and solution of a cost function $f : \{0,1\}^n \to \mathbb{R}$. In the beginning, the cost function is encoded in a quantum state. Then, Grover's search finds a negative sample, the function is shifted by the obtained value and this process is repeated until there are no more negative values, thus the minimum is reached (Figure 3.1). The scheme of this procedure is reported in Figure 3.2.

This chapter is organized in the following way: Section 3.1 explores the state preparation mechanism to encode the cost function into a quantum state and Section 3.2 discloses the algorithm in detail.

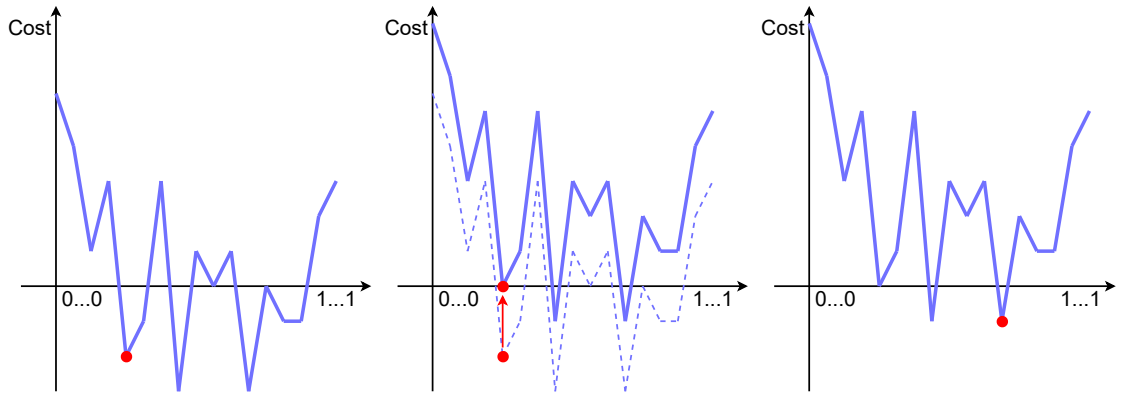


Figure 3.1: Visualization of the GAS procedure on the function profile. In the leftmost plot, a negative sample is identified. In the middle one, the function is shifted. In the rightmost one the global minimum is detected.

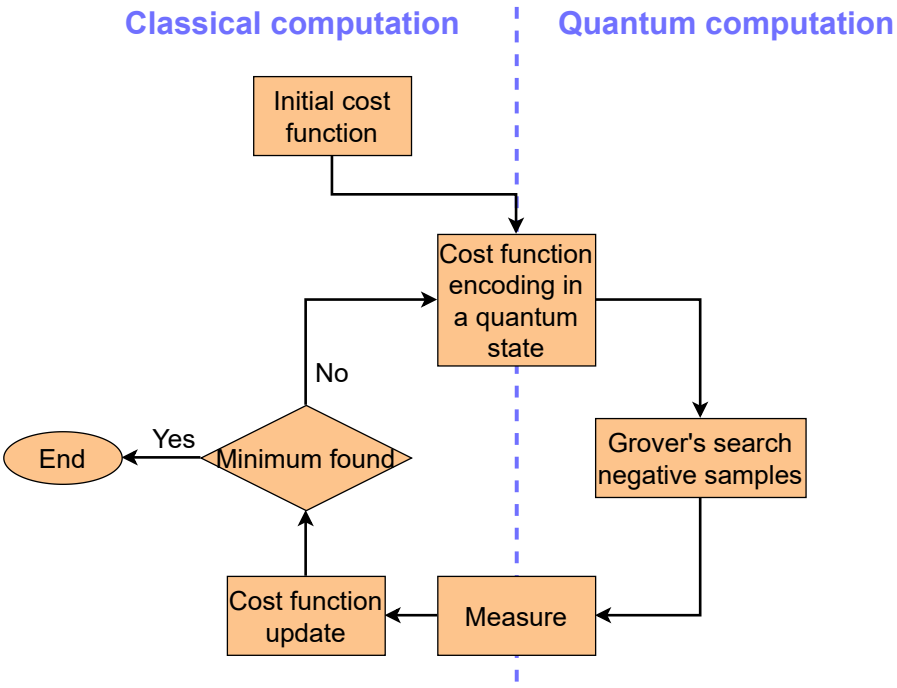


Figure 3.2: Grover adaptive search algorithm scheme

3.1 Quantum dictionary

The GAS algorithm needs an operator (here called A_y) to prepare an input state composed of an equal superposition in key-value pair format of all $|x\rangle_n$ inputs and $|f(x) - y\rangle_m$ outputs. It is called **quantum dictionary** [18] and is the quantum counterpart of a dictionary, i.e., a data structure in which values are associated with unique keys. The former are the values of the objective function corresponding to the input x shifted by the amount y, while the latter are the inputs. In this way, when a new function sample is found, it is possible to know both the value and the input configuration required for obtaining it.

$$A_y |0\rangle_n |0\rangle_m = \frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} |x\rangle_n |f(x) - y\rangle_m . \tag{3.1}$$

In order to realize the A_y operator each value has to be encoded in the Fourier basis, i.e., representing the number as a rotation across the z-axis, starting from the x-axis, of the base angle $\frac{2\pi}{2^n}$ times the number that is wanted to be encoded for the most significant qubit, then the double of the angle for the next qubit and so on. A controlled phase gate is used to rotate the state of a qubit according to the binary values of the controlled qubits, namely the keys. In this way, keys and values are entangled and when a value is found by the Grover search algorithm, the

input state will collapse to a solution associated with it. The phase gates encode each coefficient of the objective function coupling it with the variables which appear in the same term. For instance, for the term $q_{ij}x_ix_j$, the key is the configuration of variables $x_i = 1$ and $x_j = 1$ and the value is q_{ij}. Therefore, the gates have as control qubits the ones related to variables x_i and x_j and encode the value q_{ij} in the local phase angles of the value qubits, namely the Fourier basis outlined in Section 1.5. Once this process is performed for all the terms in the objective function, the state will be a superposition of entangled key-value couples composed of all the configurations of input variables and the values assumed by the function. In the end, when the superposition of all the key-value pairs is created, the inverse QFT is applied to the value register so that the encoded numbers are transformed into the usual computational basis, encoding them in the two's complement binary number representation. This is obtained thanks to its similarity with the Fourier encoding since angles up to $\pi/2$ correspond to positive numbers, while angles from $\pi/2$ correspond to negative ones just like two's complement numbers are positive if their most significant bit is 0, while they are negative, if it is 1. For instance, $7\pi/4$ is equal to $-\pi/4$ as 111_2 is equal to -001_2.

An example encoding the simple function $f(x) = -x_0x_1$ is reported in Figure 3.3 and the resulting measured states obtained by simulating the circuit are reported in Figure 3.4. It can be observed how only the four possible states are measured: the first three corresponding to the inputs 00, 01 and 10 for which the function assumes value 0, the last corresponding to the input 11 for which the function assumes value 1. The state obtained is the equal superposition of these four states, hence the probability of measuring one state is equal for all of them. In order to draw and simulate quantum circuits the Qiskit library [19] for Python has been employed.

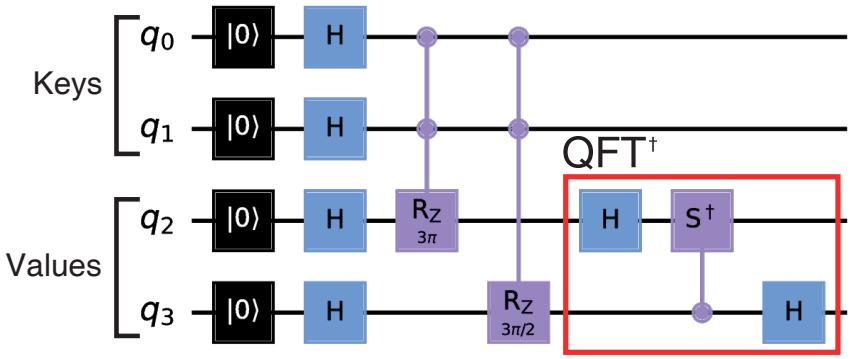


Figure 3.3: Quantum dictionary that encodes at the key $|11\rangle$ the value $|11\rangle$

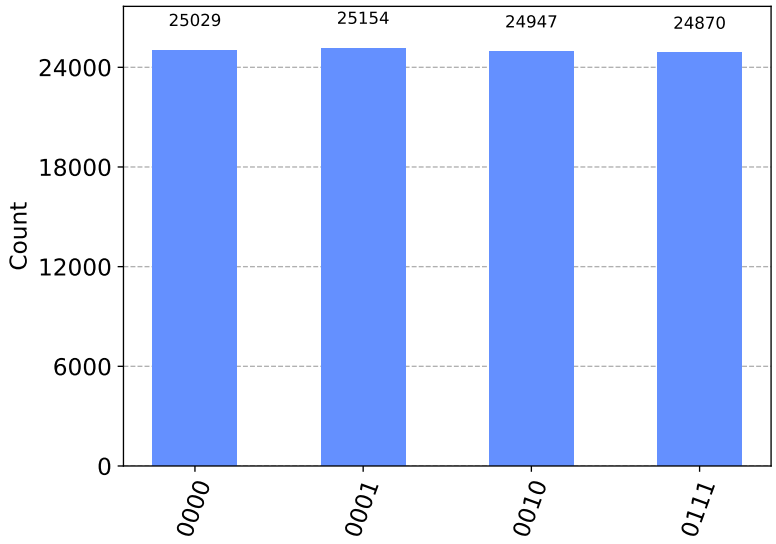


Figure 3.4: Simulation of quantum dictionary

The same initial state can be obtained by exploiting an ancilla qubit and **phase kickback** [18]. It uses controlled phase gates on the ancilla qubit to change the local phases of the states of the controlled qubits, while the ancilla remains unchanged. If a controlled gate is applied on a state, which is one of its eigenvectors, then this state is unchanged, since unitary matrices have eigenvalues with absolute value equal to 1. For example, a R_y gate, that is a rotation across the y-axis, does not affect a state lying on the same axis. Instead, for the controlled qubits, if a gate applies a global phase, this is not visible when measuring the qubit state, but if the control one is in a superposition then the phase becomes local, therefore its state can be changed. For instance, a phase gate $P(\theta)$ applies a phase shift only if the qubit is in the state $|1\rangle$, but this is a global phase. Though, if a controlled phase gate is applied to the same qubit and the control qubit is in a superposition of states [11, Sec. 2.3], the phase shift is applied only if both qubits are in the state $|1\rangle$, hence now a local phase has been applied and the state of the control qubit has changed. In this 2-qubit system example, the effect of this gate on the quantum state of the two qubits, in which the control is applied to the least significant one, is shown by the following equation:

$$\begin{aligned} {}^{C}P\left[|1\rangle \otimes \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)\right] = {}^{C}P\left[\frac{1}{\sqrt{2}}(|10\rangle + |11\rangle)\right] = \\ \frac{1}{\sqrt{2}}\left(|10\rangle + e^{i\theta}|11\rangle\right) = \left[|1\rangle \otimes \frac{1}{\sqrt{2}}(|0\rangle + e^{i\theta}|1\rangle)\right]. \end{aligned} \tag{3.2}$$

In order to obtain the same state as before, $R_y(\theta)$ controlled gates, performing a rotation of the same angle as previous gates, can be applied on the ancilla qubit,

taking as control the qubits whose value has to be assigned. As in the previous case, the inverse QFT is needed at the end to transform values from the Fourier basis to the computational basis. An example is reported in Figure 3.5 and the resulting measured states simulating the circuit are reported in Figure 3.6.

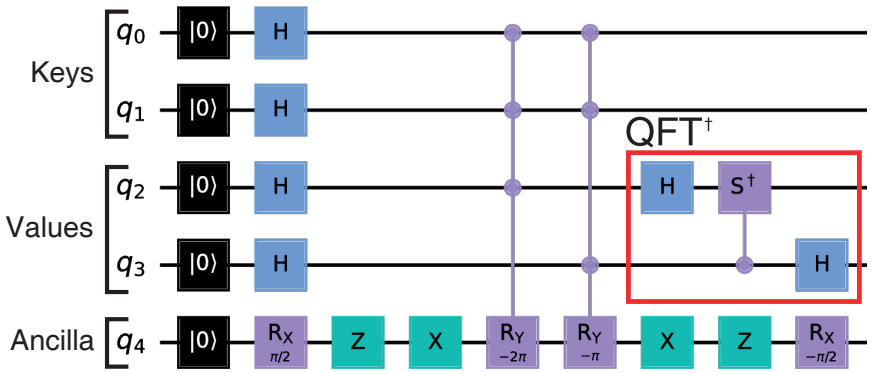


Figure 3.5: Quantum dictionary that encodes at the key $|11\rangle$ the value $|11\rangle$ using $R_y(\theta)$ gates

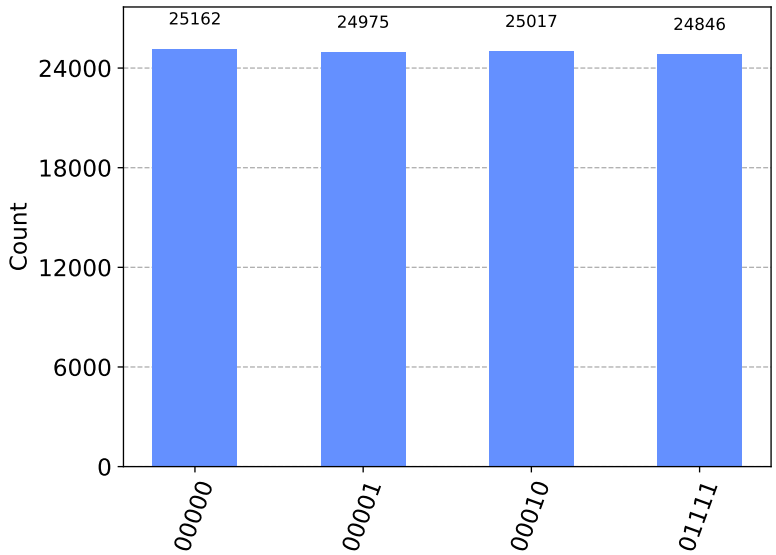


Figure 3.6: Simulation of quantum dictionary with $R_y(\theta)$ gates

3.2 Grover search of negative samples

In Section 1.4 it has been shown that Grover's search iterates the phase oracle and diffuser operators to maximize the amplitude of the state of interest, but the same

algorithm can be applied also to amplify the amplitudes of multiple states. The optimal number of quantum iterations (or rotations) r is given by the following equation:

$$r \approx \frac{\pi}{4}\sqrt{\frac{2^n}{M}}, \tag{3.3}$$

where M is the number of states of interest. This number of rotations ensures to measure one of the M states with the largest probability. However, M is usually unknown so r changes according to the cost function profile, thus GAS is employed to attain a quite accurate number of rotations. This can be achieved with a randomized approach as in the basic version of GAS or by selecting a certain policy as proposed in [20].

The two key elements of Grover's search algorithm, the state preparation and the oracle, can be designed so that all states for which the value of the function is lower than a given threshold y ($f(x) < y$) are flagged. In this way, a solution of f with a value lower than y is found and it can be used as a new threshold for the next iteration, thus finding a value closer to the global minimum until this is actually reached. Though, to have an oracle as simple as possible and equal for every iteration the threshold is always set to 0 and when a solution of f lower than 0 is found the function is shifted by that value [18]. In this way, the oracle has just to check if the value is negative and it can be done just by observing the value of the most significative qubit since numbers are represented in two's complement. The quantum circuit implementing the Grover Adaptive Search procedure is reported in Figure 3.7.

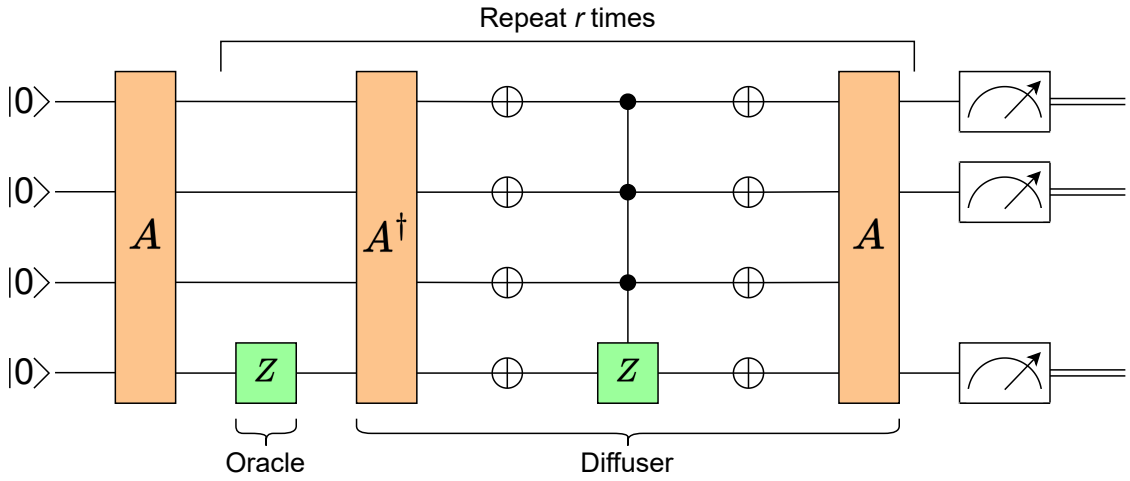


Figure 3.7: Grover Adaptive Search circuit. In this example the Diffuser performs the operation described in Section 1.4, but with a global phase -1. The result is unchanged since global phases do not affect measurements.

The state preparation, as opposed to the example shown in Section 1.4, is not

just made of Hadamard gates to create superposition, but it is made of the operators needed to construct a quantum dictionary as expressed in Section 3.1. Once the state preparation creates a superposition between inputs and function values shifted by the threshold as in Equation (3.1), the oracle performs the following operation:

$$U_f \left|x\right\rangle \left|f(x) - y\right\rangle = \text{sign}(f(x) - y) \left|x\right\rangle \left|f(x) - y\right\rangle . \tag{3.4}$$

This process (Figure 3.2) is repeated until a given number of iterations, a given elapsed time, a given value of y is reached or other termination conditions. In order to test the Grover Adaptive Search to solve optimization problems the Qiskit qasm simulator through the Qiskit library [19] has been employed.

3.3 GAS degrees of freedom

The GAS algorithm has some degrees of freedom, such as Grover rotations and the stop condition of the algorithm, that can be selected according to predefined policies. The right number of Grover rotations r, as observed in Section 1.4, ensures that the final state is as close as possible to the equal superposition of the target states. Initially, every rotation brings the state closer to its ideal destination of an angle θ, but θ depends on the number of target states, therefore, not knowing it a priori, can result in a too-small or an excessive rotation.

The ideal stop condition is when the optimal value has been sampled by Grover's search and there are no more negative values in the cost function. In this case, there is a uniform probability to sample any configuration, but sampling non-negative values can be a misleading stop condition because it can also occur when an improper number of quantum rotations has been selected. Hence, it is possible to count the number of consecutive non-negative samples and define a threshold t that stops the algorithm when it is overcome [20]. For these reasons, r has to be chosen at each iteration, since the number of labeled states changes each time the function is shifted, and t has to be determined to find a tradeoff between fast execution and a good probability of finding the global minimum. In the Qiskit implementation of the GAS solver, r is obtained randomly and increased anytime a positive value is measured, whereas the stop condition must be defined by the user. The study in [20] presents dynamic policies, i.e. that vary according to the results of each iteration, for the estimation of these two degrees of freedom.

All quantum backends have a severely limited number of qubits, which makes it very difficult to use the GAS algorithm to solve QUBO problems with a reasonably high number of variables. For exploiting the algorithm, the number of qubits employed for the variables and the function's values with the quantum dictionary must be lower or equal to the number of qubits available in a given backend. Furthermore, the number of needed qubits for function values has to be known in advance to correctly solve the problem because a lower number of qubits may lead to a

wrong result, since values will not be encoded properly, whereas a higher number of qubits will lead to a waste of resources. Therefore, preprocessing is needed to estimate an upper and a lower bound of the objective function in order to know the necessary amount of qubits. In addition, the GAS algorithm has some degrees of freedom, for which a preprocessing step can also be used to choose the best policy according to the type of problems or some parameters of the QUBO model. In this way, it is possible to use a number of Grover rotations which allows for achieving a better probability of finding a negative solution and a stop condition that permits finding the optimal solution in the shortest possible time.

Part II

Preprocessing toolchain

Chapter 4

General structure

The **preprocessing toolchain** proposed in this thesis aims to perform automatically all the steps needed to solve a combinatorial optimization problem on a solver that can be a quantum one such as quantum annealer, Grover Adaptive Search, Quantum Approximate Optimization Algorithm (QAOA) or Variational Quantum Eigensolver (VQE), or a classical one such as simulated annealing. A first step is required for translating an abstract description of the problem to solve in a cost function written in a solver-compliant formalism (QUBO in this case), eventually taking into account variable constraints. Afterwards, the other steps are not fundamental for addressing an optimization problem, but can improve the exploitability of any solver and, in particular, of quantum ones, which may not have a sufficient number of qubits. Indeed, they aim to reduce the number of involved variables, estimate the range of values assumed by the cost function and subdivide the problem into smaller ones. In this thesis, the target solver considered is the GAS algorithm and, consequently, the main effort is in reducing the number of qubits required. In particular, in order to apply the successive techniques, it is necessary to transform the QUBO problem into an **equivalent graph representation**. Exploiting the graph form, several algorithms (**fix persistencies block**) are then employed in order to reduce the number of variables, finding assignments for those variables that will be surely present in the final solution. Therefore, these variables can be removed from the graph since their value is known and the graph can be simplified. Once as many variables as possible have been removed a lower bound for the QUBO function is found once again exploiting its graph representation (**Probing block**). Thanks to the lower bound, it is possible to estimate the number of qubits needed to encode the function in a quantum dictionary.

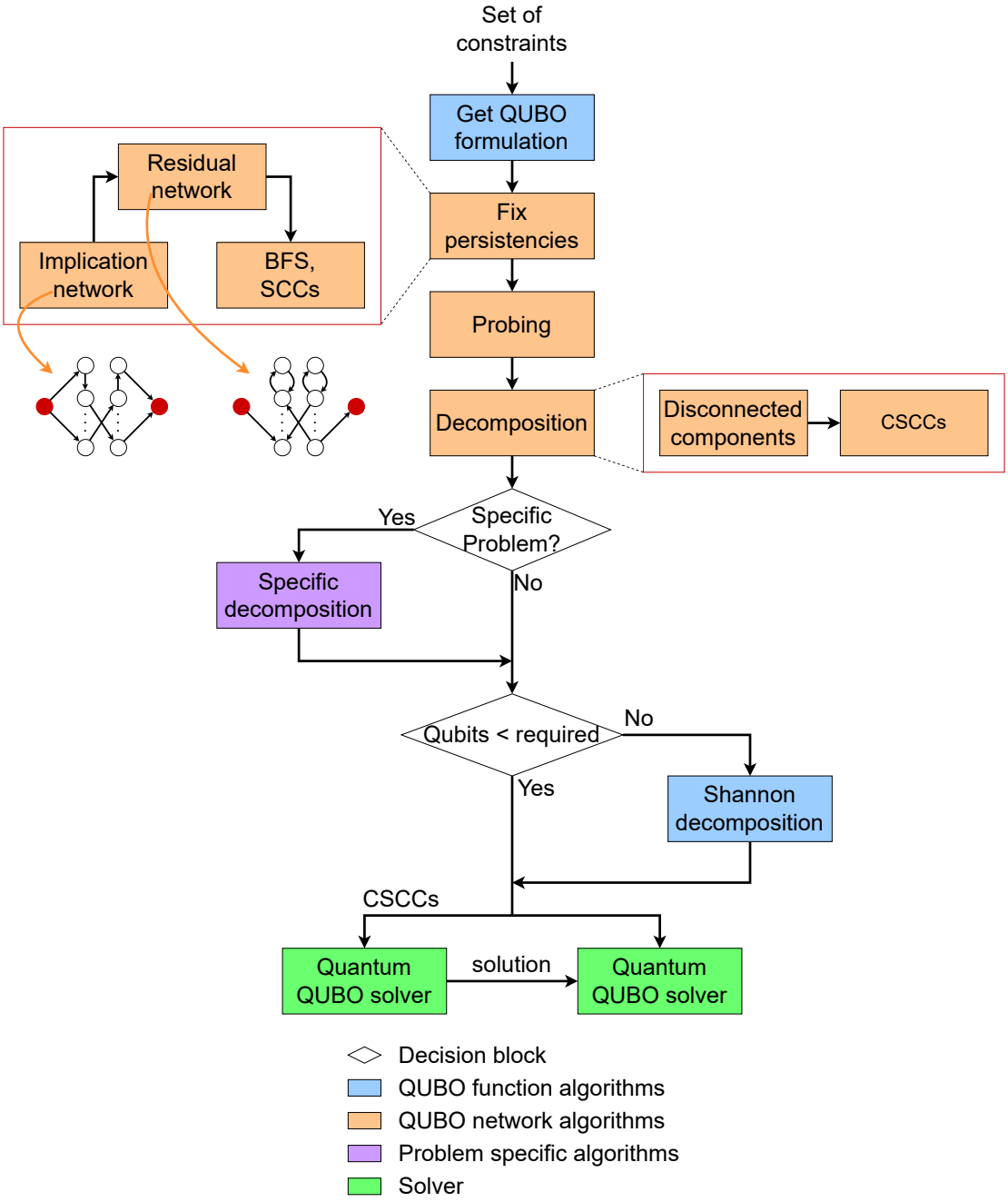


Figure 4.1: Toolchain structure

Successively decomposition techniques (**decomposition block**) acting on the graph allow splitting the graph into smaller ones so that the subgraphs will have fewer variables and it will be possible to find the minimum of the functions associated with each subgraph and then combine them to obtain the solution of the

initial QUBO problem. Furthermore, many decomposition routines have been studied for specific classes of problems, thus such routines (**specific decomposition block**) can be applied to the problem, if it belongs to one of the categories that the toolchain supports. The toolchain includes algorithms for the decomposition of graphs for maximum clique and minimum vertex cover, but it can be expanded in the future. In the end, only if all the previous methods prove to be not sufficient to reduce the number of qubits needed below the number of qubits available, another decomposition method (**Shannon decomposition block**) can be used because, although not efficient, it allows removing one variable at each iteration for any QUBO function. This last method is thought to be used just as a final resource to remove very few variables. Before using the final solver algorithms such as the Grover adaptive search, a **normalization** step could be necessary to scale down the QUBO coefficients in order to have the smallest coefficient equal to 1 and all the other ones scaled accordingly, still keeping them as integers. Finally, when all the preprocessing techniques have been performed, the QUBO functions obtained can be split into functions characterized by a strongly connected graph representation and a not strongly connected graph representation. These two groups can be solved independently by any **QUBO solver**, but for the latter group the minimum value for each function is known. The whole structure of the toolchain is reported in Figure 4.1.

The toolchain is written in C++ by using the object-oriented programming paradigm. The functions must be called by a Python program since all the interfaces for quantum solvers or quantum solvers simulators are written in this language. Hence the **Cython** language is employed to act as a bridge between C++ and Python. Cython allows writing C or C++ extensions for Python, allows static type declarations in a Python code, translates the source code into C or C++ and compiles it as a Python extension module, achieving a great speed up with respect to pure Python [21]. In this way, the programmer can use the high-level Python interface, with the possibility to write wrappers, i.e., Python functions to interface the code with C or C++ external modules, and execute code within the standard CPython environment but at the speed of compiled C or C++ language. Cython is based on **Pyrex** which is a language for writing Python extension modules hiding the Python/C API to the user, so the user can mix Python and C data structures without knowing the Python/C API [22]. In order to wrap C++ functions in a Pyrex file (or in a Python file), the header file in which the function is declared has to be included and the declaration of the function to be used has to be translated into the Cython syntax. At this point, the C++ function can be called within a wrapper function that converts parameters from Python data structures to C++ ones and converts the returned objects from C++ data structures to Python ones. Afterwards, a *setup.py* script is needed to make Cython compile a C++ source. In this file, an extension object is created to which some parameters have to be specified: the extension name, all the source files, the Pyrex file and the language

of the source files, in this case, C++. Executing this script, Cython generates a C++ file, which is the optimized translation of the wrapper, and a **shared object** file that is the library made by the wrapper and by the C++ source files which can be imported at runtime by Python files, just as a usual Python module.

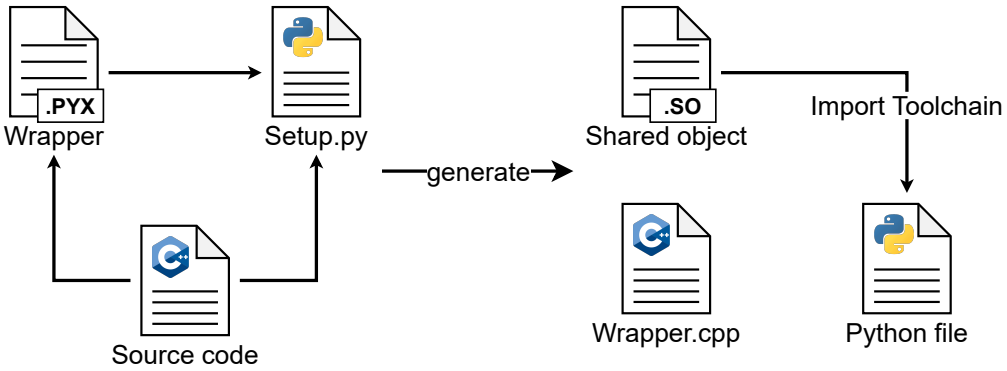


Figure 4.2: File organization for the generation of the Python extension module with Cython

Chapter 5

QUBO network representation

Any QUBO objective function has an immediate equivalent graph representation that can be obtained considering a node for each variable with a weight equal to the coefficient of its linear term and considering an edge for each quadratic relation connecting the two variables interacting, having as weight the value of the quadratic coefficient.

However, this graph form does not allow the application of preprocessing algorithms, thus, in the following sections, another graph representation valid for any QUBO model, which can be exploited, since using this particular symmetrical structure many algorithms, to achieve the presented preprocessing goals, — such as variable reduction, estimation of the lower bound of the optimum value and function decomposition — will be presented. Section 5.1 shows an alternative mathematical form of writing the cost function, Section 5.2 discloses how to transform it in the equivalent graph representation. Successively, Sections 5.3 and 5.4 explore all the graph transformations needed to apply the toolchain methods.

5.1 Posiform

For each binary variable x_i within the QUBO model, its **complement** can be defined as $\overline{x_i} = 1 - x_i$ and all the variables and their complemented counterparts are called **literals**.

Any QUBO function, or in general any pseudo-boolean function, can be represented as a **posiform**. A posiform [23] ϕ is a multilinear polynomial expression, defined on the set of all literals, that has only positive coefficients, except, eventually, the constant term.

$$\phi(x) = \sum_{T \subseteq L} c_T \prod_{u \in T} u, \tag{5.1}$$

where L is the set of the literals and $c_T \geq 0$, if $T \neq \emptyset$. The degree of the posiform is the size of the largest subset of literals for which $c_T \geq 0$.

Every quadratic pseudo-boolean function can be transformed in a quadratic posiform with linear complexity depending on its size, that is the number of its nonzero terms. Starting from the function of Equation 2.1, this transformation can be performed by substituting a variable x_i with $1 - \overline{x_i}$ for any term with a negative coefficient. Hence, for every quadratic term having $q_{ij} < 0$:

$$q_{ij}x_ix_j = q_{ij}x_i(1 - \overline{x_j}) = q_{ij}x_i + (-q_{ij})x_i\overline{x_j}. \tag{5.2}$$

Therefore, it is obtained the equation:

$$f(x) = q_0 + \sum_{i=1}^{n} c_i x_i + \sum_{\substack{1 \leq i < j \leq n \\ q_{ij} > 0}} q_{ij}x_ix_j + \sum_{\substack{1 \leq i < j \leq n \\ q_{ij} < 0}} (-q_{ij})x_i\overline{x_j}, \tag{5.3}$$

where:

$$c_i = q_i + \sum_{\substack{i < j \leq n \\ q_{ij} < 0}} q_{ij}. \tag{5.4}$$

Now for every linear term with $c_i < 0$, the substitution $x_i = 1 - \overline{x_i}$ is applied, so, in the end, it is obtained:

$$f(x) = c_0 + \sum_{\substack{i=1 \\ c_i > 0}}^{n} c_i x_i + \sum_{\substack{i=1 \\ c_i < 0}}^{n} (-c_i)\overline{x_i} + \sum_{\substack{1 \leq i < j \leq n \\ q_{ij} > 0}} q_{ij}x_ix_j + \sum_{\substack{1 \leq i < j \leq n \\ q_{ij} < 0}} (-q_{ij})x_i\overline{x_j}, \tag{5.5}$$

where

$$c_0 = q_0 + \sum_{\substack{i=1 \\ c_i < 0}} c_i. \tag{5.6}$$

Writing the result in a more compact form, the posiform is represented by the following equation:

$$\phi(x) = c_0 + \sum_{u \in L} c_u u + \sum_{u,v \in L} c_{uv}uv, \tag{5.7}$$

in which $c_u \geq 0$ and $c_{uv} \geq 0$ for any $u, v \in L$. From a quadratic posiform is also possible to go back to a quadratic pseudo-boolean function using the inverse procedure.

5.2 Implication network

An **implication network** is an equivalent representation of a quadratic posiform in the form of a directed weighted graph. In order to represent the function as an implication network, first of all, the posiform must be transformed into a purely quadratic one, thus all linear terms in the posiform are multiplied by a fictitious

variable x_0 with a constant assignment $x_0 = 1$. Therefore, all terms can be considered quadratic. Then, moving the constant term to the left-hand side of the equation, a homogeneous quadratic expression is attained:

$$\phi(x) - c_0 = \sum_{u,v \in N, u \neq v} c_{uv} uv, \tag{5.8}$$

where N is the set of literals including x_0 and $\overline{x_0}$, which coincides with the set of nodes of the implication network. The set of edges is constituted by two edges for any couple of literals (u, v) that appear in a term in the posiform, one edge between the nodes u and $\overline{v}$ and the other edge between v and $\overline{u}$, both having as weight γ half of the coefficient of that term's value.

$$\gamma_{u\overline{v}} = \gamma_{v\overline{u}} = \frac{1}{2} c_{uv}. \tag{5.9}$$

There is a one-to-one correspondence between implication networks and quadratic posiforms, thus given an arbitrary implication network, it is always possible to associate it with a quadratic posiform. The implication network is part of a specific category of graphs called **networks** [24, 25], which are directed graphs with non-negative weights in which a source node s, with only outgoing arcs, and a sink t, with only incoming arcs, are distinguished. In such networks, a **feasible flow** can be defined as a mapping $\varphi : N \times N \to \mathbb{R}$ that satisfies the following constraints:

- **capacity constraint**: an arc's flow can not exceed the weight (or capacity) of that arc:

$$0 \leq \varphi(u, v) \leq \gamma_{uv}; \tag{5.10}$$

- **conservation condition**: for every $u \in N/\{s, t\}$ the net flow across the node u is 0:

$$\sum_{e^+ = u} \varphi(e) = \sum_{e^- = u} \varphi(e), \tag{5.11}$$

  where e^+ indicates the end-point node of the arc e and e^- indicates the starting-point node of the arc e.

Networks are naturally suitable to model transportation infrastructures (Figure 5.1). The source can be seen as a production center and the sink can be seen as the destination. The weights of the edges represent the maximum rate at which goods can be transported along a certain track. In this analogy, the flow stands for the actual rate at which goods are transported on each edge, so the flow can never overcome the maximum rate of each edge (capacity constraint) and the rate at which the products enter a node must be the same at which these products leave the node (conservation condition).

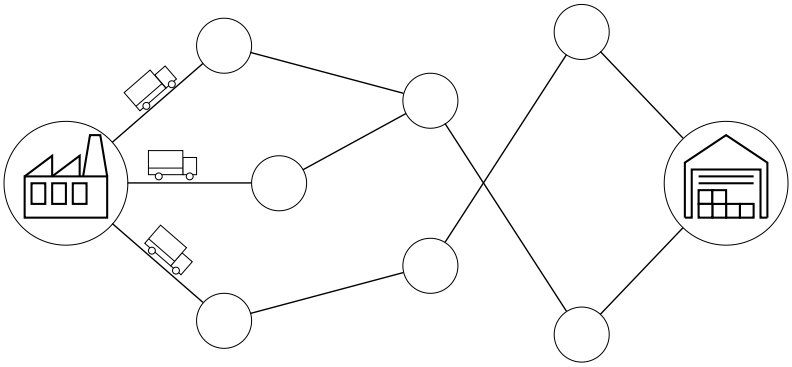


Figure 5.1: Example of network modeling a transportation infrastructure

Given a network $G(N, E)$, where N is the set of nodes and E is the set of edges, a **residual network** is a network with node set N and edge set E_φ with **residual weight** $\gamma_\varphi(e) = \gamma(e) - \varphi(e)$ defined as the difference between the edge's weight and its flow. Consider that for any edge (u, v) the flow between nodes v and u is provided by $\varphi(v, u) = -\varphi(u, v)$. The **flow value** $v(\varphi)$ is the sum of the flow of each edge leaving the source or, analogously, entering the sink:

$$v(\varphi) = \sum_{(s,v)\in E} \varphi(s, v) \tag{5.12}$$

The **maximum flow** is the particular flow for which there are no augmenting paths in the network. An **augmenting path** is a path joining source and sink in the residual network and for each edge in the path $\gamma(e) > 0$. The **maximum flow value** is the value of the maximum flow and it can be indicated with $v(G, \gamma)$. The maximum flow corresponds to the maximum rate of goods transported from the production center to the destination along any possible path.

In this thesis, the edges in the residual network going towards the source or coming from the sink will be never reported because they have no role in all the algorithms, that make use of this graph representation, presented in the next chapters.

Example. *Construct an implication network from a QUBO function.*

$$f(x) = 3 + 2x_1 + 3x_2 - x_3 + x_4 + x_1x_2 - 4x_2x_3 + x_3x_4 - 2x_4x_5.$$

First of all, the function has to be converted in a posiform, thus for every quadratic term with a negative coefficient the transformation 5.2 is applied.

$$\begin{aligned}\phi(x) &= 3 + 2x_1 + 3x_2 - x_3 + x_4 + x_1x_2 - 4x_2(1 - \overline{x_3}) + x_3x_4 - 2x_4(1 - \overline{x_5}) \\ &= 3 + 2x_1 - x_2 - x_3 - x_4 + x_1x_2 + 4x_2x_3 + x_3x_4 + 2x_4x_5.\end{aligned}$$

Now the same transformation is applied to linear terms:

$$\begin{aligned}\phi(x) &= 3 + 2x_1 - (1 - \overline{x_2}) - (1 - \overline{x_3}) - (1 - \overline{x_4}) + x_1x_2 + 4x_2\overline{x_3} + x_3x_4 + 2x_4\overline{x_5} \\ &= 2x_1 + \overline{x_2} + \overline{x_3} + \overline{x_4} + x_1x_2 + 4x_2\overline{x_3} + x_3x_4 + 2x_4\overline{x_5}.\end{aligned}$$

Multiplying all the linear terms for the fictitious variable x_0, with the constant assignment $x_0 = 1$, the posiform becomes a homogeneous quadratic polynomial

$$\phi(x) = 2x_0x_1 + x_0\overline{x_2} + x_0\overline{x_3} + x_0\overline{x_4} + x_1x_2 + 4x_2\overline{x_3} + x_3x_4 + 2x_4\overline{x_5}.$$

The implication network can be constructed by producing two edges $(u, \overline{v})$, $(v, \overline{u})$ for each term uv in the posiform. The result is shown in Figure 5.2.

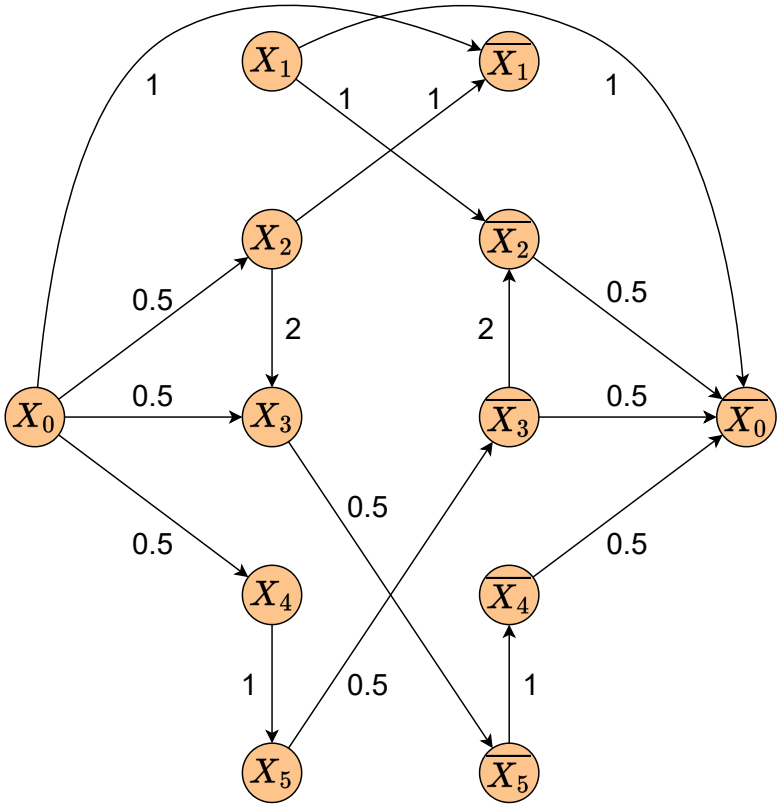


Figure 5.2: Implication network associated with the cost function $f(x)$

Note that the implication network has a symmetrical structure by construction. △

Calling Ψ the quadratic posiform derived from the residual network $G_\Psi(N, E_\varphi)$, with residual capacities γ_φ, obtained from the feasible flow φ in the implication network G_ϕ, the following equations, presented in [23], hold.

$$\phi = c_0 + v(\varphi) + \Psi. \tag{5.13}$$

$$c_0 + v(G, \gamma) \leq \min \phi(x) \tag{5.14}$$

The inequality is called **roof dual bound**. This is a lower bound for the minimum of the QUBO function that can be used to estimate the qubits needed to encode the values of the function in a quantum dictionary.

5.3 Maximum flow

The best-known way to find the maximum flow in a network is the Ford-Fulkerson algorithm, which also shows the connection between the value of the maximum flow and the minimum weight cut that disconnects the source and the sink. This method consists of finding the augmenting paths in the network. However, the Push-Relabel method, introduced by Goldberg and Tarjan [26], based on the concept of preflow, ensures finding the maximum flow with lower computational complexity. The **preflow** is a function on vertex pairs that satisfies the same properties of a flow except for the conservation constraint which is relaxed in the nonnegativity constraint. The conservation constraint for a flow, moving the right-hand side of Equation (5.11) to the left, is:

$$\begin{aligned} \sum_{\substack{u \in N/\{s,t\} \\ u=e^+}} \varphi(u,v) - \sum_{\substack{u \in N/\{s,t\} \\ u=e^-}} \varphi(v,u) = 0 \\ \implies \sum_{u \in N/\{s,t\}} \varphi(u,v) = 0, \end{aligned} \tag{5.15}$$

whereas the nonnegativity constraint for a preflow is:

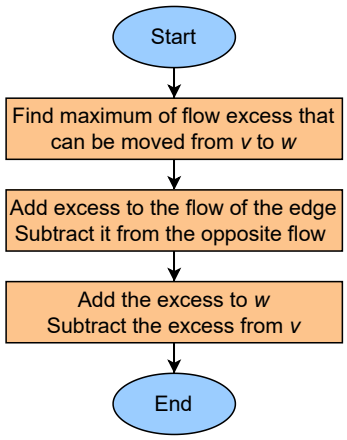
$$\sum_{u \in N/\{s\}} \varphi(u,v) \geq 0. \tag{5.16}$$

Therefore, the total flow into any vertex except s is greater than or equal to the total flow out. The **flow excess** for a vertex v is the net flow into v:

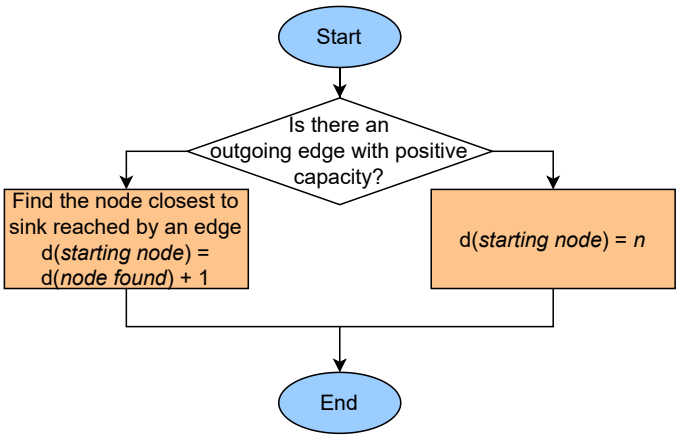
$$e(v) = \sum_{u \in N} \varphi(u,v). \tag{5.17}$$

The algorithm examines vertices different from s and t with positive flow excess. It pushes the excess to vertices closer to the sink t and if the sink is not reachable from that vertex the excess is pushed to vertices closer to the source. Once all the vertices have zero excess the preflow becomes the maximum flow. If a vertex v has positive excess and an edge (v,w) has positive residual capacity γ_φ, an amount of flow excess equal to $min(e(v), \gamma_\varphi(v,w))$ can be moved from v to w. The algorithm begins with the preflow equal to the capacity of the edge for the edges leaving the source and equal to 0 for all the other edges. In order to determine whether or not a vertex is closer to the sink or the source, the distance of a vertex from s and t is estimated by the **labeling function** $d : N \to \mathbb{N}$, which is defined such that $d(s) = n$, where n is the number of nodes, $d(t) = 0$ and $d(v) \leq d(w) + 1$ for every residual edge (v,w). The initial label for each node except source and

sink can be assigned to 0. However, a more efficient one can be computed with a breadth-first search (BFS) from the source and then from the sink. It assigns as a label to each node either the distance from the sink or the number of nodes if a node is reachable only from the source. A *push* (figure 5.3a) operation from v to w adds an amount of flow given by $min(e(v), \gamma_\varphi(v,w))$ to $\varphi(v,w)$ and $e(w)$ and subtracts the same amount from $\varphi(w,v)$ and $e(v)$. A *relabel* (figure 5.3b) operation on v sets the label of v as $d(v) = d(w) + 1$ taking the minimum $d(w)$ considering all the vertices w connected to v, but if v has no edge with positive residual capacity the label is set as $d(v) = n$ and the vertex becomes inactive. The algorithm terminates when there are no more active vertices, where a vertex v is active if $v \in V/\{s,t\},\ d(v) < n,\ e(v) > 0$.



(a) *push* operation on edge (v,w)



(b) *relabel* operation on vertex v

Figure 5.3: Flowcharts of Figure 5.3a *push* and (b) *relabel* operations

The same routine is repeated for each vertex which is selected according to an order established by a certain policy. It can perform three operations for a selected vertex v:

1. a push is performed on the current edge (v,w) if the residual capacity of the edge is larger than 0 and if $d(v) = d(w) + 1$,
2. if 1. is not applicable, if the current edge is not the last one, it changes the current edge to the next edge going out from v,
3. if 1. and 2. are not applicable, it makes the first edge the current one and it applies a *relabel* operation

The current edge of a vertex v is defined as the current candidate for a *push* operation and it is, initially, the first edge outgoing from v. The *discharge* (Figure 5.4) operation repeats this routine until the excess on that vertex is null or the

label of that vertex is increased and at the same time it keeps track of the order of the vertices to be selected.

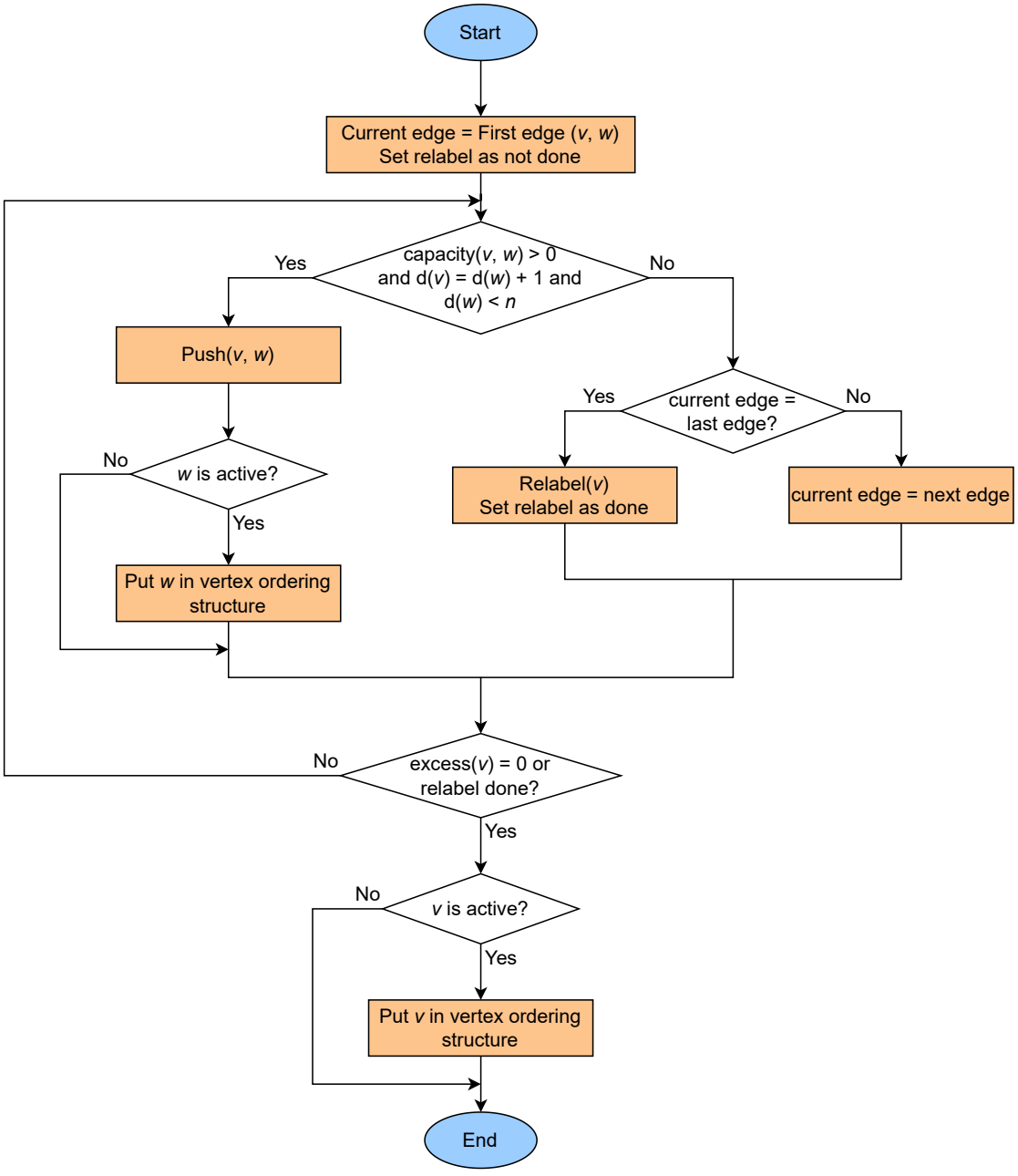


Figure 5.4: Flowchart of *discharge* operation on vertex v

To select the order of the nodes to be processed, a FIFO queue can be employed. At every iteration, a vertex is removed from the queue and the *discharge* operation is repeated until the queue is empty. If a vertex becomes active during these steps, it

is added to the queue. By using a FIFO queue, the algorithm has $O(n^3)$ complexity, where n is the number of nodes in the network. The **highest-level** (**HL**) **selection strategy**, instead, has $O(n^2\sqrt{m})$ complexity, where m is the number of edges [27]. Therefore, this is the procedure adopted in the toolchain. This strategy selects the node with positive excess furthest from the sink, so the one with the highest label. In this way, every push operation will be performed on a node with maximal flow excess, which means that the subtree rooted at this node has no other flow excesses. Thanks to it, the total number of pushes is decreased [28]. The implementation of this strategy uses an array in which for each index is stored a list of all the active nodes that have a label equal to that index and an index b that is the highest label among the active nodes. At each iteration, a node is taken from the list at the index b of the array, it is processed using the *discharge* operation and b is updated.

This algorithm allows finding the value of the maximum flow, but in order to obtain the residual network a second step, that transforms the preflow in a flow, is needed. To achieve this result, the nodes with positive excess have to be processed in **reverse topological order**. A topological order is a linear succession of nodes such that for each edge (u, v) in the graph, the node u comes before the node v in the succession [29]. A reverse topological sort algorithm can be realized using a **postorder depth-first search** (DFS) on the transposed graph, which is a graph that has the same edge set but every edge has opposite direction. In order to obtain a postorder traversal, nodes are added to the list keeping track of the order when the DFS last visits them. A topological sort is possible only for directed acyclic graphs, thus if the DFS finds a cycle, the flow is decreased until one of the edges in the cycle has 0 flow and then the search is restarted. A cycle of flow has to be removed since it is not contributing to transport flow from the source to the sink. Following this reverse order, when a vertex with positive excess is being processed it is ensured that all the vertices that are reachable from itself have already been visited. Therefore, its excess can no longer increase. This means that for each node, all the possible flow is pushed back to them from nodes closer to the sink before they are visited. In this way, they can be visited just once ensuring that the flow is always going back towards the source and the visited nodes are always left with 0 excess and it will never increase again. Repeating this process for all the nodes following the reverse ordering until the source, all the excesses of the visited nodes become null, hence the preflow becomes a flow.

Example. *Calculate the maximum flow on a network.*
The maximum flow on the implication network of the example in Section 5.2 must be calculated to obtain the residual network. As a starting point, the initial labeling function is calculated and the result is reported in Figure 5.5. The nodes are visited with a BFS starting from the sink and the label is set as the distance, in terms of edges traversed, from the sink. Nodes that do not have a path leading to the sink are labeled with the number of variables. The first step of the algorithm moves the

flow from nodes with positive excess to nodes closer to the sink alternating *push* and *relabel* operations until the maximum flow value is found.

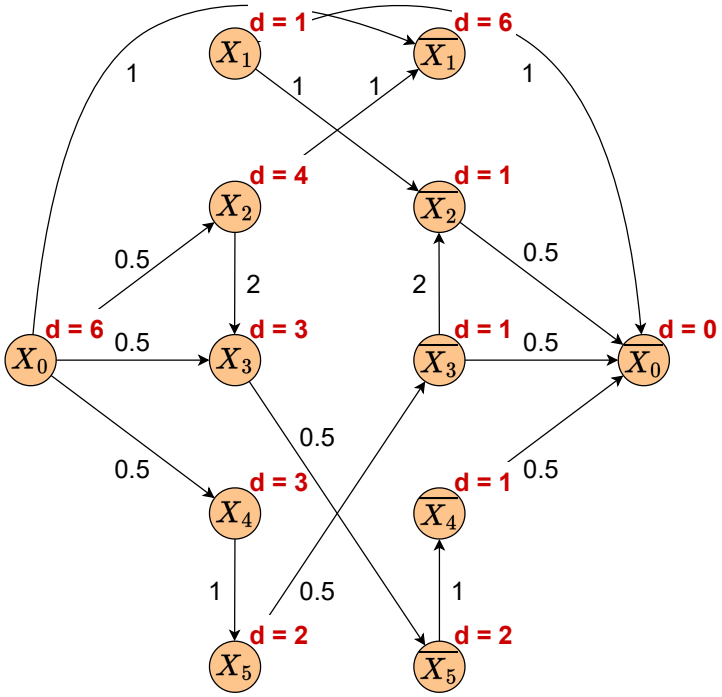


Figure 5.5: Initial labeling function

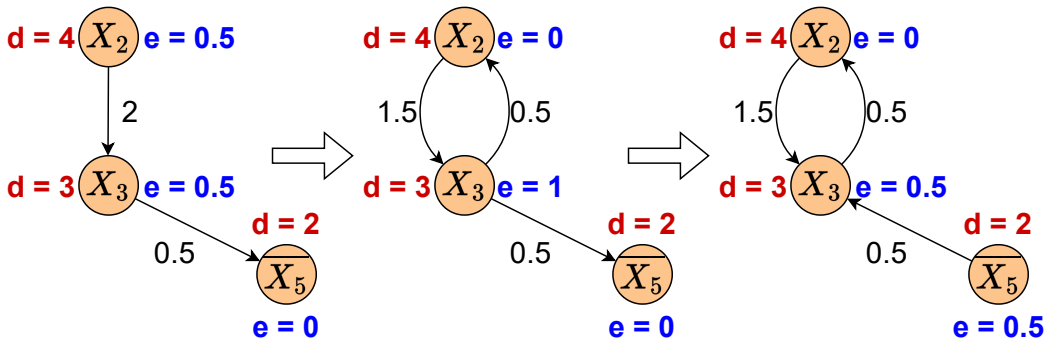


Figure 5.6: Example of the push-relabel method first step

Figure 5.6 shows what the algorithm does in a portion of the graph composed of the nodes x_2, x_3 and $\overline{x_5}$. The preflow value of the edges leaving the source is initialized at the capacity of these edges. x_3 is closer to the sink than x_2, thus a flow equal to the excess of x_2 is pushed to x_3.

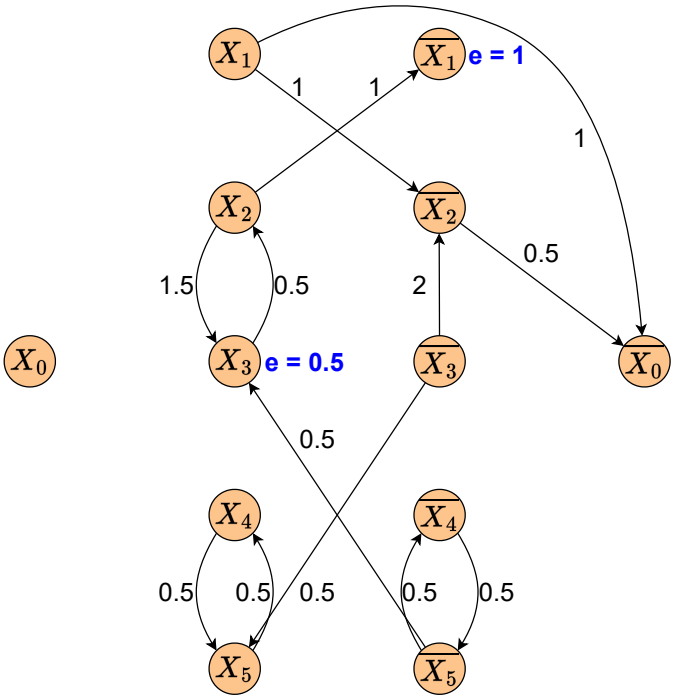


Figure 5.7: First step of push-relabel method

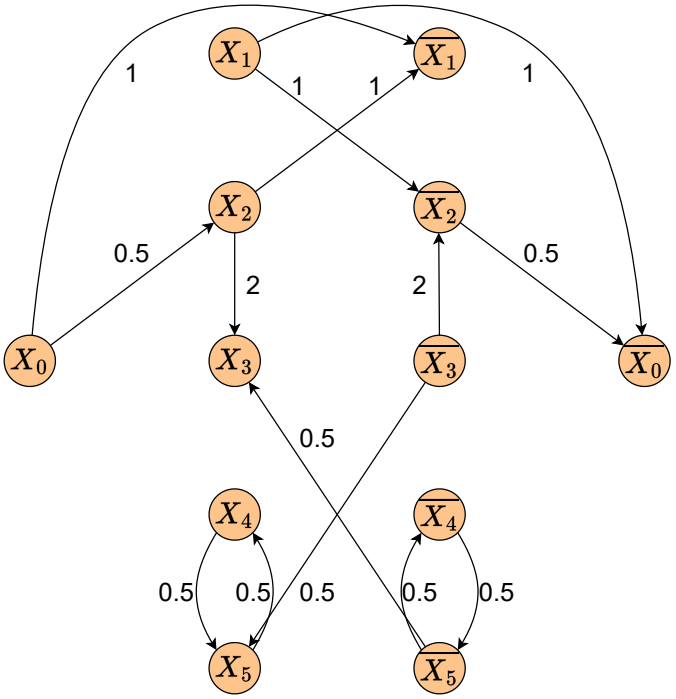


Figure 5.8: Residual network

Afterward, the same condition occurs with x_3 and $\overline{x_5}$, but this time the capacity of the edge is lower than the excess, hence a flow equal to the capacity is pushed to $\overline{x_5}$. The final result is shown in Figure 5.7.

The maximum flow value can be calculated as the sum of the flow going into the sink, but it can be seen that the flow is still a preflow since some nodes still have an excess greater than 0. For this reason, in the second step, all the excess remaining in the nodes is pushed back to the source. Therefore nodes $\overline{x_1}$, x_2, x_3 are analyzed in reverse topological order obtained from the starting network, that is $(\overline{x_1}, x_3, x_2)$ and finally the residual network is achieved (Figure 5.8). △

5.4 Residual network

Calculating the maximum flow on the implication network, the residual network can be constructed, but another step may be needed before applying the preprocessing techniques. There is a one-to-one correspondence between the posiform and its associated implication network and in order to keep this correspondence between the posiform and its associated residual network the structure of the latter must be symmetric. If an edge (u, v) has positive residual capacity, the edge $(\overline{v}, \overline{u})$ must exist and must have exactly the same capacity as the first edge. The algorithm disclosed in Section 5.3 correctly calculates the value of the maximum flow and a residual network, but it does not ensure that the produced network is symmetric, thus it has to be made symmetrical. For each edge (u, v) in the residual network, the final weight is given by the average between the weight of the edge and its symmetrical counterpart:

$$\gamma_\varphi(u, v) = \gamma_\varphi(\overline{v}, \overline{u}) = \frac{1}{2} \left(\gamma_\varphi(u, v) + \gamma_\varphi(\overline{v}, \overline{u})\right). \tag{5.18}$$

Example. *Symmetrization of a residual network.*
Given an objective function:

$$f(x) = 4 + 4x_2 - 4x_1x_2 + 4x_1x_3 - 4x_2x_3\,,$$

and computing the associated posiform, then implication network and finally the maximum flow using the push-relabel method (Figure 5.9), the value of the maximum flow is correct. It can be seen that there are no augmenting paths in the network. However, the obtained graph does not respect the symmetry imposed by the one-to-one correspondence with the posiform. Edge (x_1, x_2) has not the same weight as edge $(\overline{x_2}, \overline{x_1})$. Therefore, the coefficients of the posiform can not be computed with $c_{ij} = 2\gamma_{\varphi(i,\bar{j})} = 2\gamma_{\varphi(j,\bar{i})}$ as in Equation (5.9) because the equality

is not true. Therefore, the residual network can be symmetrized with the process described above, and observing the result in Figure 5.10, it is possible to check that the value of the maximum flow is still 2, it has not changed, and it is still maximum because there are no augmenting paths. The maximum flow function is not unique, thus another one has been found that makes the residual network symmetric and it will be used to apply all the preprocessing techniques.

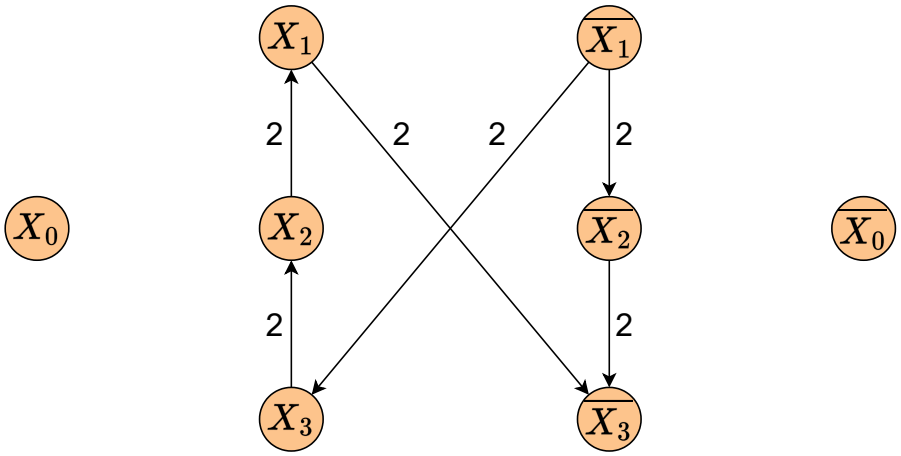


Figure 5.9: Residual network obtained with the push-relabel method

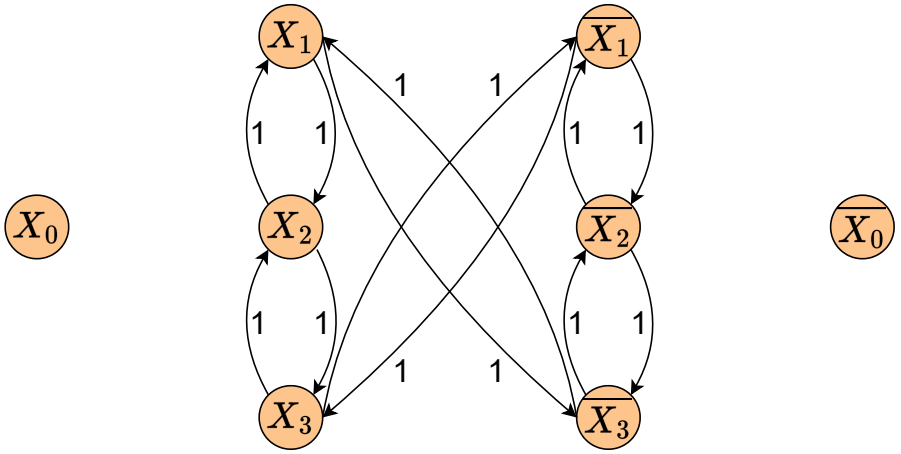


Figure 5.10: Residual network after symmetrization operation

△

Chapter 6

Persistencies

In this chapter, methods to reduce the size of a QUBO cost function are explored. Sections 6.1 and 6.2 detect nodes that can be removed in the residual network by exploiting two different graph algorithms, whereas Section 6.3 applies temporary changes to the network to pinpoint additional removable nodes and also obtains a lower bound for the function minimum.

Let x_i be a binary variable in a pseudo-boolean objective function and let $a \in \mathbb{B}$ be a binary value, x_i is a strong (or weak) **persistency** [23], if $x_i = a$ for all minima (or at least one minimum) of the objective function. Finding a persistency means individuating the value a for a certain variable, hence it can be substituted in the function reducing the number of variables. The algorithms presented in this chapter aim to find as many persistencies as possible from the QUBO model, reducing the number of qubits needed to encode the function in a quantum solver.

6.1 Source-reachable persistencies

Let $S \subseteq N$ be the set of nodes $v \in N$, in the residual network constructed with the maximum flow φ, for which it exists a directed path from the source x_0 to the node v that has every arc with positive capacity. Moreover, let $T = \{\overline{v} \mid v \in S\}$. It is impossible to have $\{u, v\} \in S$, if there is a quadratic term $a_{uv}uv$ with $a_{uv} > 0$. Otherwise, it would be present in the residual network a positive capacity arc from u to $\overline{v}$, where $\overline{v} \in T$, so it is connected to $\overline{x_0}$, thus violating the maximality of the flow. Recall that a flow is maximum if there are no augmenting paths, i.e., paths connecting the source to the sink. Therefore, all the literals in S are strong persistencies since the assignment that sets all these literals to 1 is present in every solution because it makes vanish all the terms including literals belonging to S. By construction of the implication network, a linear term $a_u\overline{u}$ in the posiform produces an edge (x_0, u) in the network. Hence, if the literals $u \in S$, directly connected with x_0, are assigned to 1 the linear terms will vanish. Furthermore, the nodes $\overline{v} \in S$

that are connected to u form a quadratic term, but, as previously stated, $v \notin S$, so v does not appear in a linear term. Consequently, if $\overline{v}$ is assigned to 1, the quadratic terms will vanish too. To sum up, if $u \in S$ and there is an arc $(u, \overline{v})$, then $v \notin S$ otherwise there would be an augmenting path. This makes it possible to find the optimal values of all the variables present in S. To identify these persistencies, the toolchain uses a **breadth-first search** (**BFS**) starting from the source that marks all the visited nodes. For this reason, they are called source-reachable. This algorithm explores all the paths starting from the source, thus detecting all the literals in S. Since all the terms containing the persistencies vanish, the residual network is updated eliminating all the nodes corresponding with them. Finally, these variables and their assignments are stored and they will be joined to the rest of the solution to achieve the final result.

Example. *Detection of source-reachable persistencies.*
Consider the cost function of the examples in Section 5.2 and 5.3:

$$f(x) = 3 + 2x_1 + 3x_2 - x_3 + x_4 + x_1x_2 - 4x_2x_3 + x_3x_4 - 2x_4x_5.$$

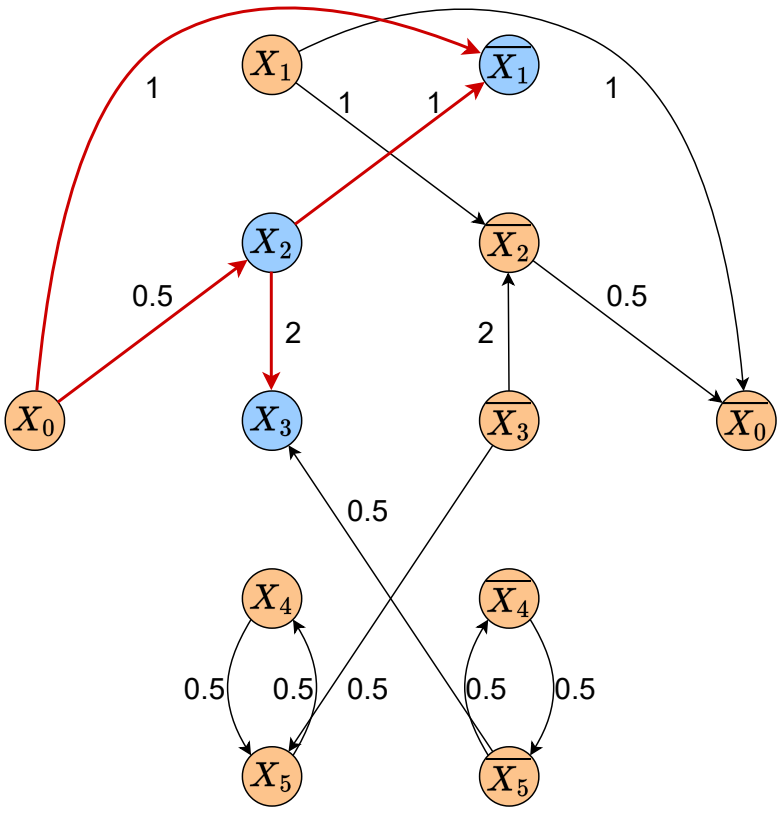


Figure 6.1: Residual network with source-reachable persistencies highlighted in blue and paths from the source colored in red

In Section 5.3, the residual network has been derived (Figure 6.1), so in this graph, a BFS is executed starting from the source x_0. All the nodes visited, i.e., that can be reached from the source, are persistencies and can be assigned to 1. They are highlighted in Figure 6.1.

In this case, the nodes x_2 and x_3 and the complemented node $\overline{x_1}$ have been visited, therefore the variables x_2 and x_3 are assigned to 1, whereas x_1 is assigned to 0. △

6.2 Strongly connected persistencies

The **strongly connected components** K_i (also called **strong components**) of a graph are the subgraphs composed of a set of nodes for which each node can be reached by any other one (example in Figure 6.2). In a residual network, it holds that either:

- $\{\overline{v} \mid v \in K_i\} \in K_i$;
- $\{\overline{v} \mid v \in K_i\} \in K_{i'}$;

where $K_{i'}$ is a set that contains the complements of the literals present in K_i and is a strong component as well because of the symmetry property of such networks.

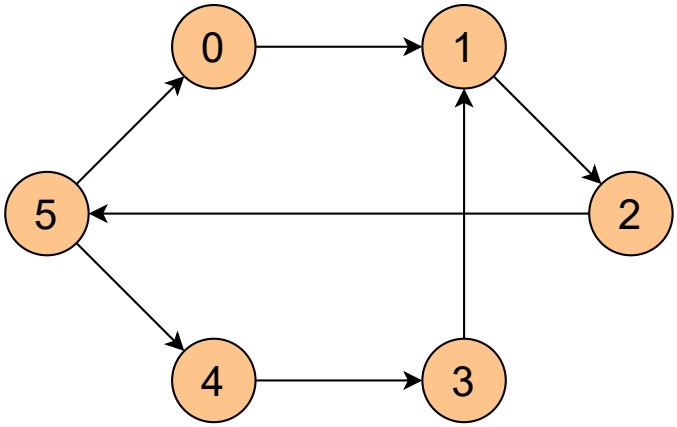


Figure 6.2: Example of a strongly connected directed graph

Strong components K_i, in which for every literal v also $\overline{v}$ is present, will be called **complete strongly connected components** (**CSCCs**) and they will be treated in Section 7.2. For the strong components K_i that imply the existence of twin strong components $K_{i'}$, the variables contained are all weak persistencies. Therefore, the assignment that sets all the nodes $v \in K_i$ to 1 and the assignment that sets all the nodes $v \in K_i$ to 0 are present in at least two different optimal

solutions of the cost function. Indeed, if the literals in the strong components have all the same value, the terms including them will vanish, thus not increasing the posiform value. In an implication network, an edge $(u, \overline{v})$ symbolizes a term uv in the posiform, and to make it vanish if u is equal to 1, $\overline{v}$ must be equal to 1, while if $\overline{v} = 0$, u must be equal to 0. Hence, literals in a strong component must all have the same value.

If there is a directed path in the residual graph from the source x_0 to K_i, or if there is an edge between K_i and $K_{i'}$, the variables contained are strong persistencies. In the former case, the persistencies are source-reachable, therefore it is already handled by the BFS algorithm, as described in Section 6.1. In the latter case, if the edge starts in K_i and goes to $K_{i'}$, the literals in $K_{i'}$ must equal 1 and the literals in K_i must equal 0 since they are complementary. In an implication network, a node equal to 1 can only lead to nodes equal to 1. Hence literals in K_i can not be equal to 1 otherwise literals in $K_{i'}$ should be equal to 1 contradicting the complementarity of the components. Vice versa, if the edge starts in $K_{i'}$ and goes to K_i, the literals in K_i must equal 1 and the literals in $K_{i'}$ must equal 0. Since these persistencies are found in a strong component, they are called strongly connected.

In order to determine strongly connected persistencies, the first step is to identify the strongly connected components in the residual network. This is achieved through **Tarjan's algorithm** [30] that is treated in Section 6.2.1. Once these components are identified, strong persistencies assignments can be stored to add them to the final solution and the corresponding variables can be eliminated by the network since their value is known in all the optimal solutions. On the other hand, before removing weak persistencies, a routine is needed to avoid the choice of assignments belonging to different optimal solutions, thus giving origin to a non-optimal one. In an implication network, if the value of a variable depends on the value of another one, there is a path that connects the latter to the former, whereas if a variable has no connections, its value does not affect the assignments of the other variables. Therefore, if a component of weak persistencies is not connected to any other weak persistencies, its literals can be indistinctly assigned to 0 or 1. The two different assignments are present in two different, but both optimal, solutions. Instead, if this subgraph is reached by a path starting from other weak persistencies, the variables can not be assigned to any value because they depend on the other ones. This scenario can be detected by launching a reverse BFS from one of the literals in the component that finds whether there are other nodes, associated with persistencies, that can reach the starting literal. Hence, in this case, the variables are temporarily not assigned because when the subgraph of weak persistencies which can reach them is processed, it is possible to run a BFS starting from a node of this component, assigning to 1 all the nodes reached by the graph traversal algorithm. In this way, also all the other components of persistencies that have a dependency on the group of literals under examination can be assigned.

Example. *Detection of weak strongly connected persistencies.*
Take the same cost function of the previous example and the already obtained residual network to which the source-reachable persistencies have been removed. At this point, the toolchain launches Tarjan's algorithm to identify the strongly connected components. However, it can be clearly seen by inspection that the two sets of nodes $\{x_4, x_5\}$ and $\{\overline{x_4}, \overline{x_5}\}$ form two strong components (Figure 6.3). Since there is no connection with the source and among them, the variables x_4 and x_5 are weak persistencies. Therefore, there is one solution where $x_4 = x_5 = 0$ and another one where $x_4 = x_5 = 1$. In this example, the techniques employed have been able to solve the optimization problem, even without resorting to a final quantum or classical solver. The two final solutions are:

$$x_1 = 0, x_2 = 1, x_3 = 1, x_4 = 1, x_5 = 1;$$
$$x_1 = 0, x_2 = 1, x_3 = 1, x_4 = 0, x_5 = 0.$$

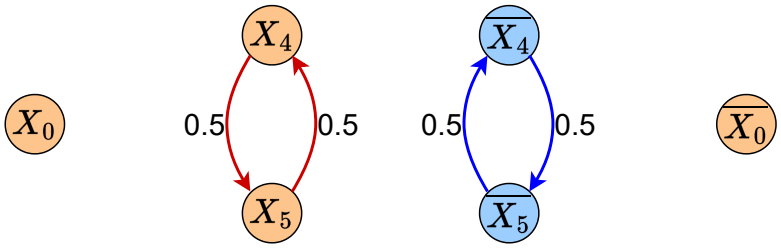


Figure 6.3: Residual network with the two strong components highlighted

△

Example. *Detection of strong strongly connected persistencies.*
Now consider a new cost function:

$$f(x) = 4 + 4x_2 - 4x_1x_2 + 2x_1x_3 - 4x_2x_3.$$

The corresponding posiform can be easily calculated, obtaining:

$$\phi(x) = 4\overline{x_1} + 4x_1\overline{x_2} + 2x_1x_3 + 4x_2\overline{x_3}.$$

Following the rules expressed in section 5.2 the implication network in Figure 6.4 is attained.

Computing the maximum flow, the residual network in Figure 6.5 is found. In this network, the two strongly connected components are highlighted with red and blue colors. It can be seen that they are connected by two edges $(\overline{x_3}, x_1)$ and $(\overline{x_1}, x_3)$, thus the variables in these components are strong persistencies. If $x_1 = 0$, then the

node $\overline{x_1} = 1$, but this would imply that also $x_3 = 1$. x_1 and x_3 are part of the same strong component, hence also $x_1 = 1$, but this violates the initial hypotesis. Therefore, the presence of the two edges between the strong components allows inferring that $x_1 = x_2 = x_3 = 1$ is a strong persistency, so it is the only optimal solution.

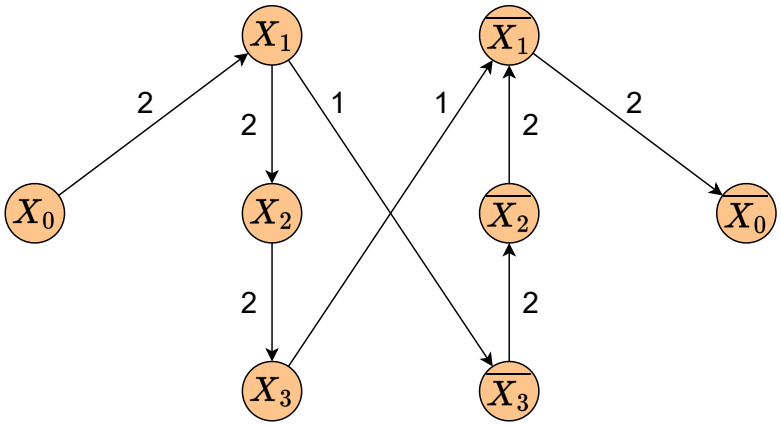


Figure 6.4: Implication network

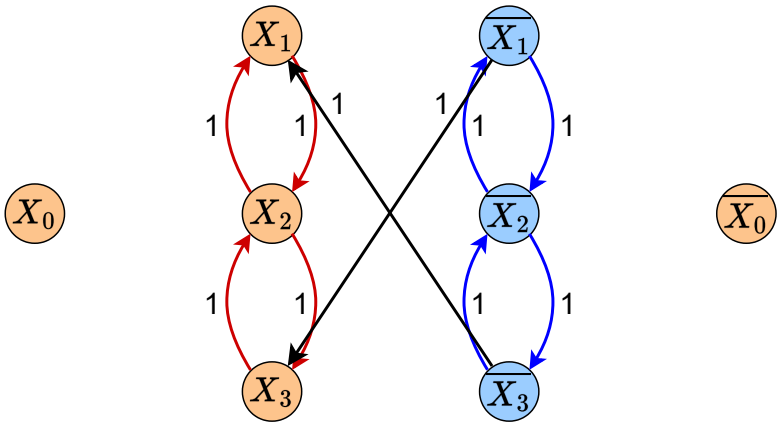


Figure 6.5: Residual network highlighting the two strong components

△

6.2.1 Tarjan's strongly connected components algorithm

An algorithm for finding strongly connected components with computational complexity $O(V, E)$ in any directed graph $G(V, E)$ was proposed by Tarjan in [30]. The order of the visits performed by the **depth-first search** (**DFS**) on a directed graph

generates a **tree**, whose edges are the ones explored when a new node in the graph is discovered. The node-set is given by the visited nodes. In this tree, also edges that go from a node to one of its ancestors, i.e. nodes that have been discovered before the current one, can be considered. These edges can be part of two categories: edges that go from descendants back to ancestors, called **fronds**, and edges that connect a subtree to another subtree, called **cross-links**. To keep the order of when nodes of the graph are first visited, each one is associated with the **discovery number**, which starts at 1 for the node from which the search is started and is incremented for every visited node. The strongly connected components form a subtree in which from all the nodes it is possible to reach the root of the subtree. Therefore, the problem of finding strong components is equivalent to finding the roots of the subtrees in the DFS tree. If two nodes are part of a strongly connected component, they have the highest common ancestor, that is the one that has been first discovered, that also belongs to the component. This is because at least a frond or a cross-link is needed to have a closed path that allows connecting the two nodes passing through the ancestor. To keep note of the highest ancestor of every node, the **lowlink** of a node v is defined as the vertex, with the lowest discovery number, reachable from a subtree having v as the root. The lowlink number of a node is the discovery number of its lowlink. Hence, the root of a strong component can be identified whenever the discovery number of a node is the same as its lowlink number since no frond or cross-link generates a path from this node to another one with a lower discovery number.

Example. *Detection of strongly connected components in a generic graph.*
Consider the graph shown in Figure 6.6, where the nodes are already numbered according to the order in which they are visited using a DFS.

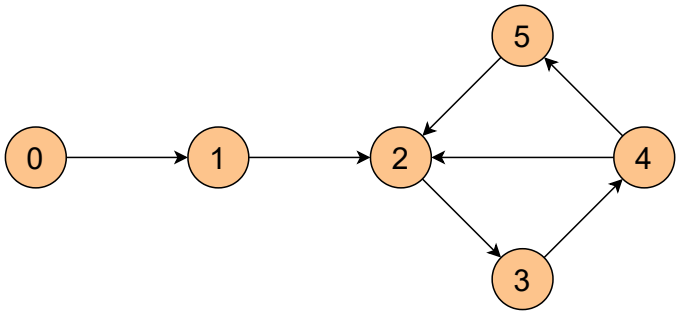


Figure 6.6: Example graph

Therefore, the DFS produces the tree displayed in Figure 6.7, in which the fronds are represented by dashed lines. At the top of the nodes, it is reported the lowlink number for each of them, which is the discovery number of their highest ancestor.

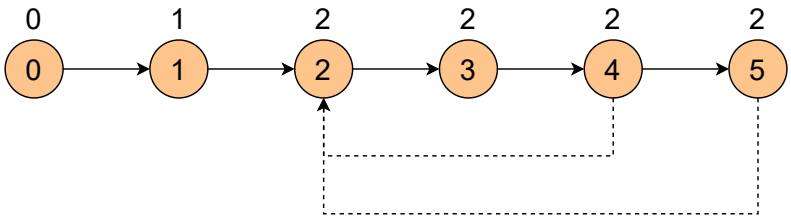


Figure 6.7: DFS tree with fronds

By inspecting the tree and the lowlink numbers it can be seen that there are three strongly connected components in the graph, consisting of the subsets of nodes $\{0\}$, $\{1\}$ and $\{2, 3, 4, 5\}$. △

The DFS can be slightly modified with the goal of calculating the lowlink for each node and identifying the strongly connected components. Every node is inserted in a stack when it is first visited. Then, for any node u, the lowlink can be computed according to two possibilities:

- if there is an edge (u, v) and v has not been already visited the lowlink number of u is given by

$$\text{lowlink}(u) = \min(\text{lowlink}(u), \text{lowlink}(v)); \tag{6.1}$$

- if there is a frond (u, v), hence v has already been visited the lowlink number of u is given by

$$\text{lowlink}(u) = \min(\text{lowlink}(u), \text{discovery}(v)). \tag{6.2}$$

  Cross-link must not be considered since they are back edges but they do not lead to closed paths in the graph. Therefore a back edge must have the endpoint in the stack to be considered.

As soon as a node with equal discovery and lowlink numbers, i.e. a root, is found, all the nodes in the stack are popped until the root is obtained. All the extracted nodes are part of a strongly connected component. The process terminates when all the nodes have been examined.

The flowchart of the modified DFS algorithm is reported in Figure 6.8. It is made the distinction between visiting and discovering a node. The former term is used to indicate when all the edges of the node have already been examined, thus all the paths starting from this node have been explored and will never be processed again. The latter term is used to indicate when a node is encountered for the first time during the graph traversal.

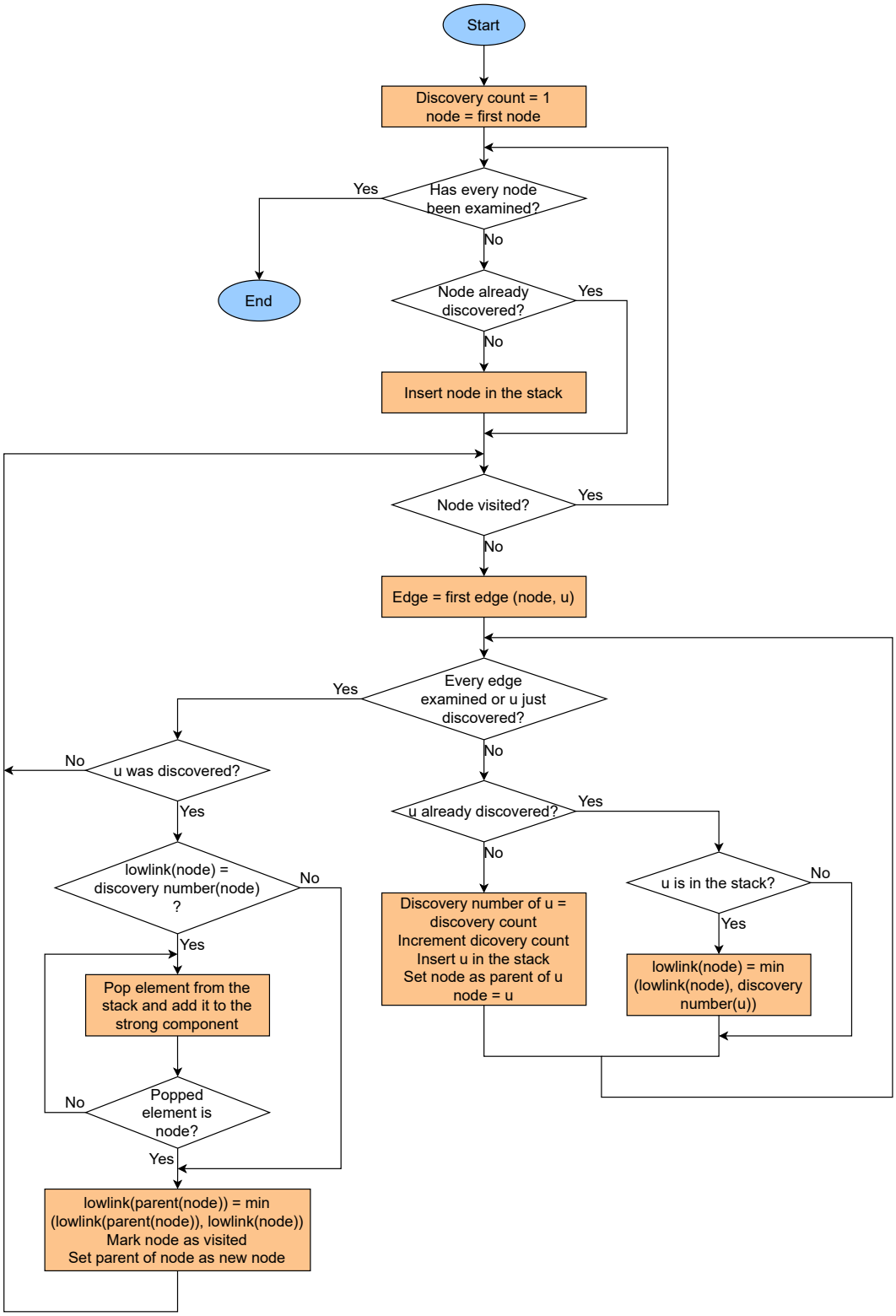


Figure 6.8: Flowchart of Tarjan's strongly connected components algorithm

6.3 Probing

The **Probing** technique modifies the residual network with the goal of finding new persistencies and a lower bound for the minimum of the function. It consists in fixing at 1 or 0 the value of one of the n variables of the cost function [23], for instance x_i. However, instead of substituting the value and evaluating the function, it generates two networks: one with an added penalty term for $x_i = 0$ and the other with a penalty for $x_i = 1$ by inserting edges to the residual network. Calling M the penalty coefficient, the following equation holds:

$$\min \phi(x) = \min(\min \phi(x) + Mx_i,\ \min \phi(x) + M\overline{x_i}). \tag{6.3}$$

The M coefficient must be large enough to ensure that the solution violating the constraint imposed on the variable x_i is higher than an upper bound on the minimum of the posiform. This is necessary to ensure that it is possible to calculate the best maximum flow, i.e. the one for which the roof dual bound (Equation (5.14)) is equal to the actual minimum. To estimate a penalty coefficient M, a method that computes an upper bound U on the minimum is needed. If there is the ideal chance of calculating the best possible flow, and if

$$M > U - c_0, \tag{6.4}$$

where c_0, is the constant coefficient of the posiform, then the edges exiting the source and entering the sink are able to carry more flow than the maximum, therefore it is possible to calculate it. Since in the toolchain Probing is directly applied on the residual network, $c_0 = 0$. Consequently, it is sufficient that $M > U$. To achieve an upper bound, the **Devour Digest Tidy-up** (**DDT**) algorithm [31] is employed. It is used the one-pass version [32] that allows converging to a solution in the fastest way, assigning a value to one variable at each iteration. In this way, by modifying the residual network adding two edges, the maximum flow can be recomputed. The roof-dual bound sets a lower limit for the minimum of the function that depends on the maximum flow. Therefore, if the flow increases the roof-dual bound gets closer to the actual minimum, thus attaining a better estimation. The Probing procedure can be repeated for all the variables, hence calculating $2n$ times the maximum flow. Calling LB_u the roof-dual bound related to the posiform $\phi(x) + M\overline{u}$, and $LB_{\overline{u}}$ the one related to the posiform $\phi(x) + Mu$, a new lower bound is obtained in the following way:

$$LB = \max_{u \in L} \left(\min(LB_u, LB_{\overline{u}})\right), \tag{6.5}$$

where L is the set of literals. Therefore, for a certain variable x_i, if the maximum flow increases both in the case of the penalty for $x_i = 0$ and in the case for $x_i = 1$, the lower bound can be increased. In Section 6.3.2, it is shown that, by modifying the residual network with Probing, it is also possible to identify new persistencies.

Example. *Improving lower bound with Probing.*
Take for example the following cost function:

$$f(x) = 4 + 2x_2 + 2x_4 - 2x_1x_2 + 2x_1x_3 - 2x_2x_3 - 2x_3x_4 + 2x_3x_5 - 2x_4x_5\,.$$

After computing the associated posiform, constructing the implication network and calculating the maximum flow, the residual network in Figure 6.9 is obtained.

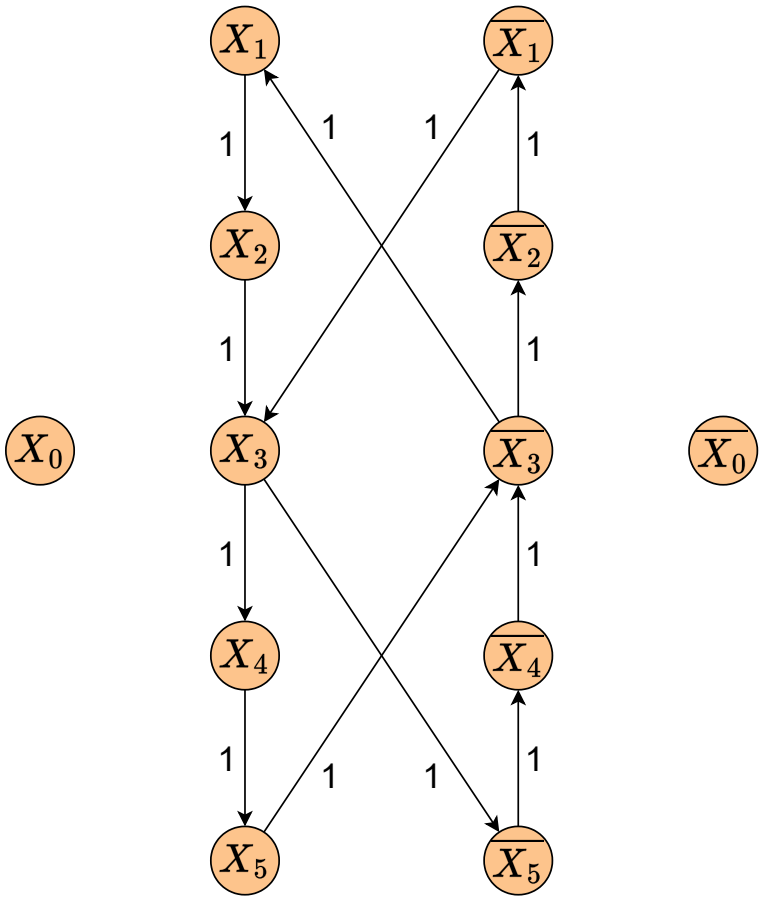


Figure 6.9: Example residual network

Now applying the Probing technique to the variable x_3 means generating two networks: one with an added term Mx_3 and one with an added term $M\overline{x_3}$. In Figure 6.10, both the couples of additive edges are reported in the same residual network to display that in both cases new paths from source to sink are formed. Therefore, the maximum flow can be increased in both cases, leading to an enhancement of the lower bound. In blue, it is highlighted the path from source to destination for the residual network associated with the posiform $\phi(x) + M\overline{x_3}$,

whereas red is used for the counterpart associated with the posiform $\phi(x) + Mx_3$. Penalty terms are represented by dashed lines.

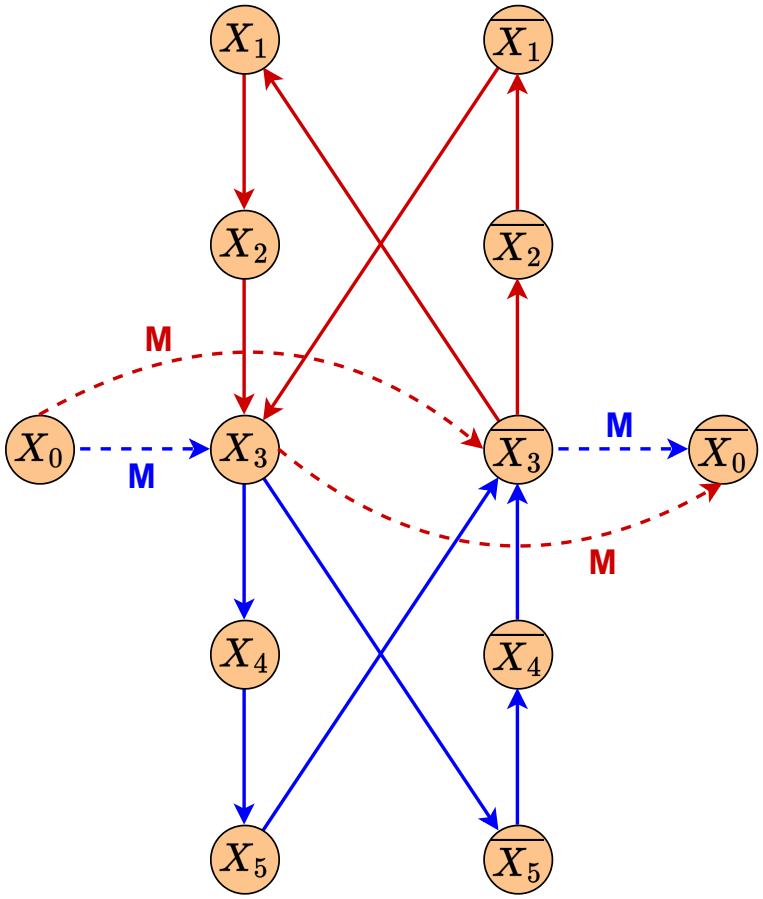


Figure 6.10: Residual network with added penalty terms for $x_3 = 0$ (dashed blue lines) and for $x_3 = 1$ (dashed red lines)

△

6.3.1 DDT one-pass heuristic

The **Devour Digest Tidy-up** (**DDT**) is a heuristic method able to provide an upper bound for the optimal solution of a QUBO problem. The solution obtained may not be the best estimation. However, a one-pass strategy reads input data just once and one-at-a-time assignments ensure that at least one variable is assigned to a binary value at each iteration. Therefore, this procedure finds a solution greater or equal to the optimal one with polynomial complexity. It can be applied to the cost function, to the posiform or to alternative representations of the posiform [32].

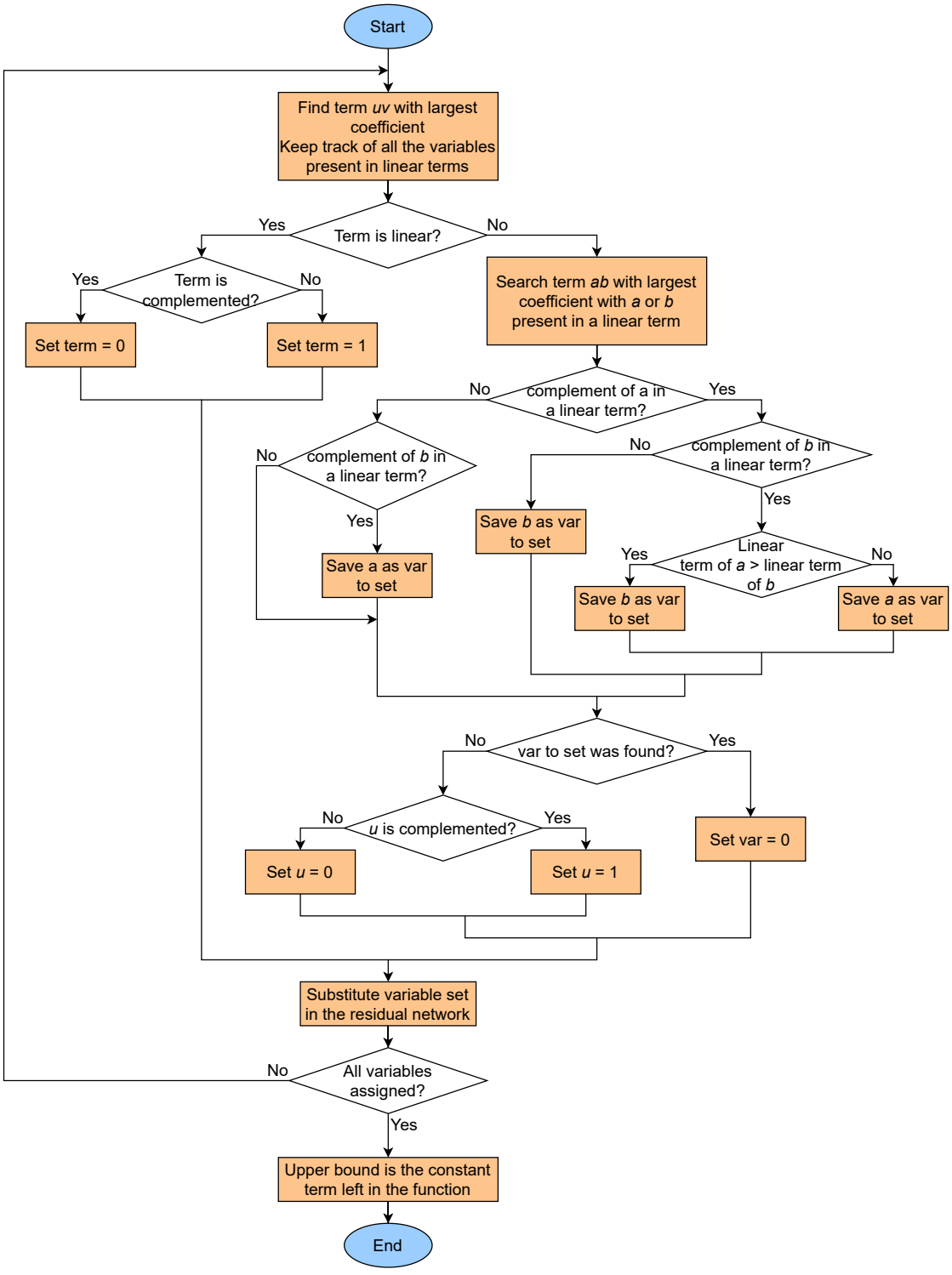


Figure 6.11: Flowchart of DDT algorithm for a residual network

In the toolchain, it has been realized a modified version that receives as input the residual network associated with a cost function, after all the persistencies have been fixed. In this way, it provides the right penalty to be added to the edges according to the Probing technique in the residual network. As the name suggests, the algorithm is divided into 3 steps:

- Devour, in which the term with the largest coefficient in absolute value is identified and is set to 1 or 0 in order to make it vanish.
- Digest, in which logical conclusions, i.e., binary assignments for variables in the term, are derived to make it vanish.
- Tidy-up, in which the previous conclusions are substituted into the residual network (or posiform in the standard case).

To implement the one-at-a-time assignments policy, in the Digest step, at least one variable is always set to a binary value. The Devour step looks for the term with the largest coefficient. If it finds a linear one, then the variable is assigned to the binary value that makes the term vanish. Otherwise, it looks for the quadratic relation with the largest coefficient formed by variables that are also present, with their complement, in a linear term. If it is found, the variable not in the linear term is set to the binary value which makes the quadratic relation vanish. In this way, the function value does not increment and the linear term can be eliminated in a successive iteration. If both are present in a linear term, the one with the lowest coefficient is chosen to make the quadratic relation vanish. In this way, the function value is incremented by the lowest amount between the two linear coefficients. Finally, if it is not found, the first variable of the term obtained at the beginning is set to the binary value that makes it vanish (flowchart in Figure 6.11).

6.3.2 Probing persistencies

The Probing technique generates two networks in which the maximum flow can be recomputed. This means that by modifying the edges with respect to the original network, it may be possible to run again the persistency-finding algorithms and identify new variables assignments that could not be previously found. Of course, the optimal value of the variable chosen for Probing is not known. Therefore, it is not known which modified network can be considered correct. For this reason, to identify new persistencies, the information provided by both networks must be combined: Given the posiform $\Psi_u = \Phi + M\overline{u}$ and defining S_u and W_u as the sets of strongly and weakly persistent literals and L_u the roof dual bound [23]:

- If $L_u > U$, $u = 0$ is a strong persistency for Φ;
- if $v \in S_{x_j} \cap S_{\overline{x_j}}$, $v = 1$, is a strong persistency for Φ;

- if $v \in W_{x_j} \cap W_{\overline{x_j}}$ $v = 1$, is a weak persistency for Φ;
- if $v \in S_{x_j}$ and $\overline{v} \in S_{\overline{x_j}}$ then $x_j = v$ is a strong persistency for Φ;
- if $v \in W_{x_j}$ and $\overline{v} \in W_{\overline{x_j}}$ then $x_j = v$ is a weak persistency for Φ;
- for all $v \in S_{x_j}$ and $w \in S_{\overline{x_j}}$ the quadratic relations $x_j \leq v$, $\overline{x_j} \leq w$ and $\overline{w} \leq v$ are all strong persistencies for Φ;
- for all $v \in W_{x_j}$ and $w \in W_{\overline{x_j}}$ the quadratic relations $x_j \leq v$, $\overline{x_j} \leq w$ and $\overline{w} \leq v$ are all weak persistencies for Φ.

Only linear relations are considered for the time being, but quadratic ones can be easily added in the future. Weak persistencies found in this way can be considered in the same way as strong ones. In a single iteration, the weak persistencies are in accordance with each other because they are found with the method described in Section 6.2. At the same time, for the other iterations, if a new weak persistency is found, it is independent of the previously detected ones. This is because, if they were correlated, there would have been an implication in the network, that is a path that connects the previous persistency with the new one, thus allowing detection of it. Therefore, a new persistency is independent of the previous one or it would have been found together with them. Consequently, the weak persistencies can all be substituted in the network with the values found at the end of the Probing process.

Chapter 7

Decomposition

Decomposition methods allow subdividing the optimization problem into smaller ones with the goal of solving subproblems with smaller sizes, i.e. with a lower number of variables, and reducing the overall execution time. Sections 7.1 and 7.2 deal with residual networks, obtaining subnetworks that can be transformed into functions that can be solved independently. In Section 7.3, routines for the decomposition of particular classes of problems exploiting their characteristics are described. Finally, Section 7.4 discloses a decomposition technique applicable to the cost function of any QUBO problem.

7.1 Trivial decomposition

The **trivial decomposition** method consists just of identifying whether the cost function is composed of subfunctions independent of each other. If a function is made up of two subfunctions:

$$f(x) = g(x) + h(x), \tag{7.1}$$

where g and h contain disjoint sets of variables then:

$$\min f(x) = \min g(x) + \min h(x). \tag{7.2}$$

Remembering that after the persistencies detection, the cost function is represented by the residual network, it is possible to identify if there are subfunctions containing disjoint sets of variables by observing whether there are **disconnected components** in this network. These components are subgraphs that are not connected by any edge. If in the residual network, two subgraphs A and B are not connected by any edge, the associated cost function is composed of terms with variables only present in A and others with variables only present in B. To find disconnected components, it is employed the disjoint-set union-based algorithm. At first, it considers every node as a disjoint set, namely using the so-called **disjoint-set data**

structure, and then unites them if they are part of the same subgraph. This procedure is detailed in Section 7.1.1

Example. *Disconnected components in a residual network.*
Consider the simple cost function

$$f(x) = x_1x_2 + x_3x_4.$$

It has no linear terms, therefore the corresponding implication network is already a residual network and it is shown in Figure 7.1.

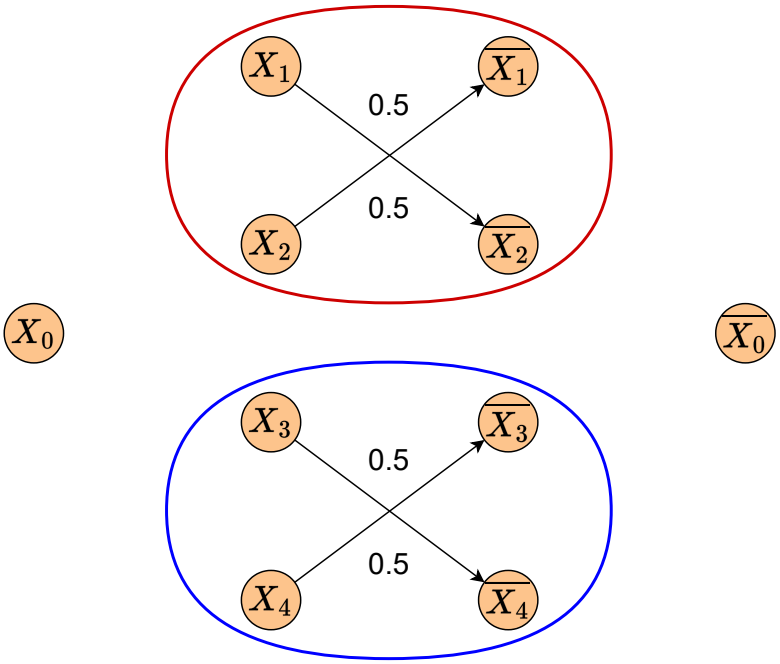


Figure 7.1: Example residual network

In the red and blue circles, two disconnected components are highlighted. Hence, the network can be divided into two subnetworks that can be solved independently. This can be clearly seen, in this example, just by looking at the function expression. The terms x_1x_2 and x_3x_4 have no variables in common and to obtain the minimum value both terms must vanish. For this purpose, it is sufficient that at least one between x_1 and x_2 equals 0 and at least one between x_3 and x_4 equals 0. △

7.1.1 Disconnected components algorithm

The algorithm explored in this section allows for finding disconnected components in undirected graphs. However, it can be used also for directed graphs, such as

networks, just by considering any edge as if it did not have a direction. When traversing a graph, if there is an edge between a and b, the node a is defined as the child of b and the node b is defined as the parent of a. The **ancestor** of a node in a connected subgraph can be defined as the node that has only children and it is the parent of itself, namely, it has no parent. The goal is to find a common ancestor for each node in a connected component so that all the nodes constituting each subgraph are identified. Therefore, finding the common ancestors means solving the problem. In the beginning, every node is initialized as the ancestor of itself, hence each one is part of its own set. Successively, for each edge, the ancestor of an endpoint is assigned as the ancestor of the ancestor of the other endpoint. In this way, all the nodes, children of the ancestor of the second endpoint, can trace back to the new, just-assigned, ancestor. Hence, when the process is terminated, from each node is possible to arrive at the common ancestor of their component. Finally, all the nodes with the same ancestor are joined in the same set. The flowchart of the algorithm is displayed in Figure 7.2.

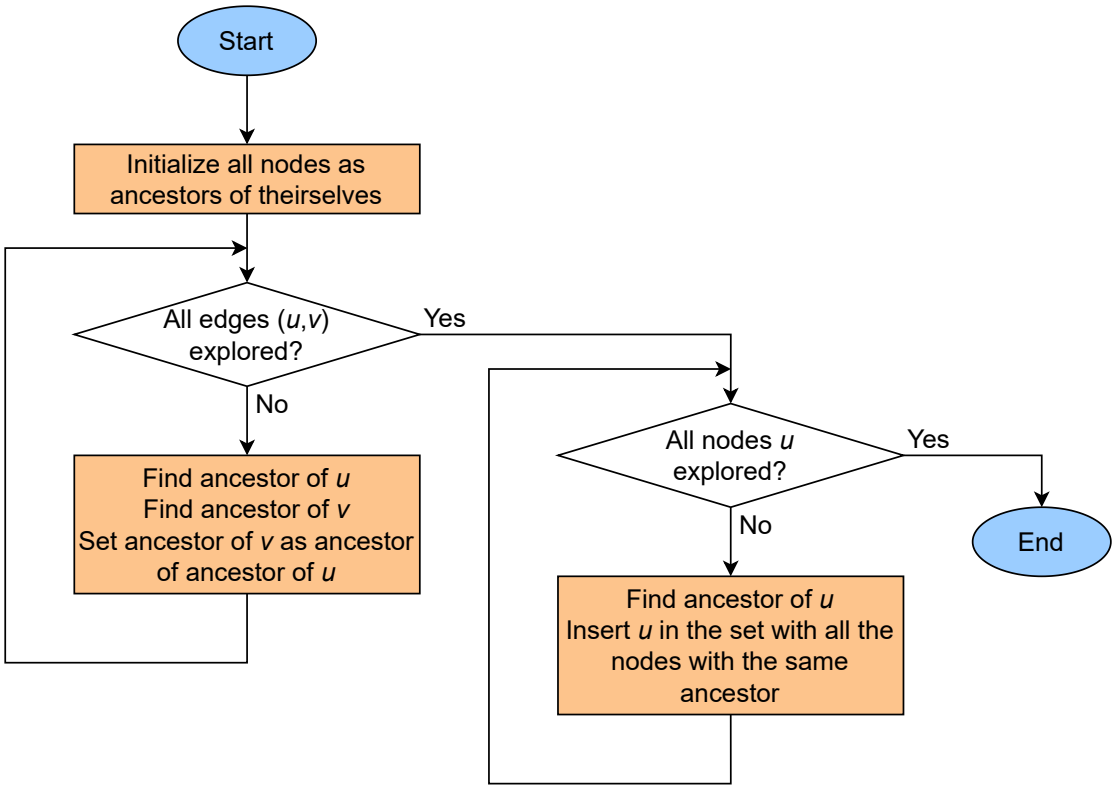


Figure 7.2: Disconnected components algorithm flowchart

7.2 Strong components decomposition

Section 6.2 shows how Tarjan's strongly connected components algorithm allows the identification of these components that can be exploited to find strong and weak persistencies, which are then removed from the residual network. The remaining subgraphs are the so-called **complete strongly connected components** (**CSCC**). Aspvall, Plass and Tarjan demonstrated in [33] that a conjunction of boolean clauses, in which each clause is a disjunction of at most two literals, is satisfiable if and only if in the implication graph associated with the boolean expression of these clauses there are no complete strongly connected components. A conjunction of boolean clauses has the form $uv+wz$, while a term in a posiform has the form $a_{uv}uv$, thus the only difference is the presence of weights, but the theorem is applicable also for pseudo-boolean functions. Hence, in a posiform, the terms can all vanish, thus not incrementing the value of the posiform, only if there are no CSCCs in the corresponding residual network. An assignment of truth values to the vertices of an implication graph makes all the terms vanish if and only if the following two conditions hold:

- a vertex x_i and its complement $\overline{x_i}$ get complementary truth values;
- no edge has starting vertex assigned true and ending vertex assigned false.

This is intrinsic in the definition of implication graph because two edges $(u, \overline{v})$ and $(v, \overline{u})$ are added for a term uv in the posiform. Therefore, for an edge $(u, \overline{v})$ the posiform has the term uv and if u is assigned to 1 and $\overline{v}$ is assigned to 0 the term vanishes. If a vertex u is in the same strongly connected component of its negated counterpart $\overline{u}$, any truth assignment to the vertices of the graph violates one of the two conditions expressed above. Since there is a path from u to $\overline{u}$, either the two vertices have complementary truth values or there is a path from a true vertex to a false one. By induction, if a vertex is marked true, it leads only to true vertices and if it is marked false it is reached only by false vertices, thus if no complete strongly connected component is present the boolean expression is satisfiable. Translating it in the QUBO context, if no complete strongly connected component is present in the residual network, the minimum value of a homogeneously quadratic posiform is 0. Recall that after all the source-reachable persistencies have been fixed, any QUBO problem is represented by a purely quadratic posiform.

If the residual network has c CSCCs and the posiform associated with each of them is indicated as ϕ_i, whereas the whole posiform is indicated as ϕ, the following relation holds:

$$\sum_{i=0}^{c} \phi_i \leq \phi, \tag{7.3}$$

which implies:

$$\sum_{i=1}^{c} (\min \phi_i) \leq \min \phi. \tag{7.4}$$

The minimum of the posiform associated with a CSCC can be determined by summing the weights of the edges with the smallest weight that if removed would make the component no longer strongly connected, which is equivalent to solving the QUBO problem corresponding to this CSCC. A posiform ϕ with CSCCs can be defined as the sum of a posiform g with no CSCCs and T_i terms, which, by adding the corresponding edges in the residual network, generate c CSCCs, with i going from 1 to c. Given a vector $x*$ for which $g(x*) = 0$, in the worst case the value of the posiform ϕ is 0 plus the weight of the edges that have been added [34]. As a consequence:

$$\phi(x*) \leq \sum_{i=1}^{c} \sum_{k \in T_i} a_k, \tag{7.5}$$

where a_k are the coefficients of the added terms. In the worst case, all the added terms contribute to increasing the value of the posiform, but:

$$\sum_{k \in T_i} a_k = (\min \phi_i), \tag{7.6}$$

implying that:

$$\min \phi \leq \sum_{i=1}^{c} (\min \phi_i). \tag{7.7}$$

Therefore, this proves that the minimum of a posiform is equal to the sum of the minima of the posiforms associated with the complete strongly connected components in the residual network. Equivalently, if the graph has c CSCCs:

$$\min \phi = \sum_{i=1}^{c} (\min \phi_i). \tag{7.8}$$

Moreover, $\phi(x*) = \min \phi$, so an assignment vector that results in the minimum value of the function can be determined from a vector that minimizes the posiform without CSCCs and the vectors which minimize the posiforms associated with CSCCs. It means that the residual network can be decomposed extracting its CSCCs and removing from the original network the edges with the minimum weight that make a component strongly connected. It is possible to know which edges have to be removed by finding the minima of the posiforms associated with the CSCCs, therefore solving the minimization problem for those posiforms. Since posiforms associated with CSCCs have to be solved first, the assignments found are directly substituted in the remaining CSCC-free posiform so that it can be solved having the lowest possible number of variables.

Furthermore, for a CSCC-free posiform or for a posiform whose CSCCs have been extracted the minimum value is known a priori. Therefore, for such functions, a **Grover oracle** can be specifically designed to find the optimal value. Consequently, the Grover Adaptive Search is no more necessary, and **Grover's search**

can be directly exploited to solve the optimization problem. This results in a significant reduction of the execution time since the solver employed is a purely quantum one, no classical iterations are needed.

Example. *Complete strongly connected components decomposition.*
Consider the following cost function to optimize:

$$f(x) = 4 + 8x_1 + 10x_2 + 4x_3 - 12x_1x_2 + 4x_1x_3 - 10x_2x_3 - 4x_3x_4 + 2x_4x_5.$$

After computing the associated posiform, constructing the implication network and calculating the maximum flow, the residual network in Figure 7.3 is obtained.

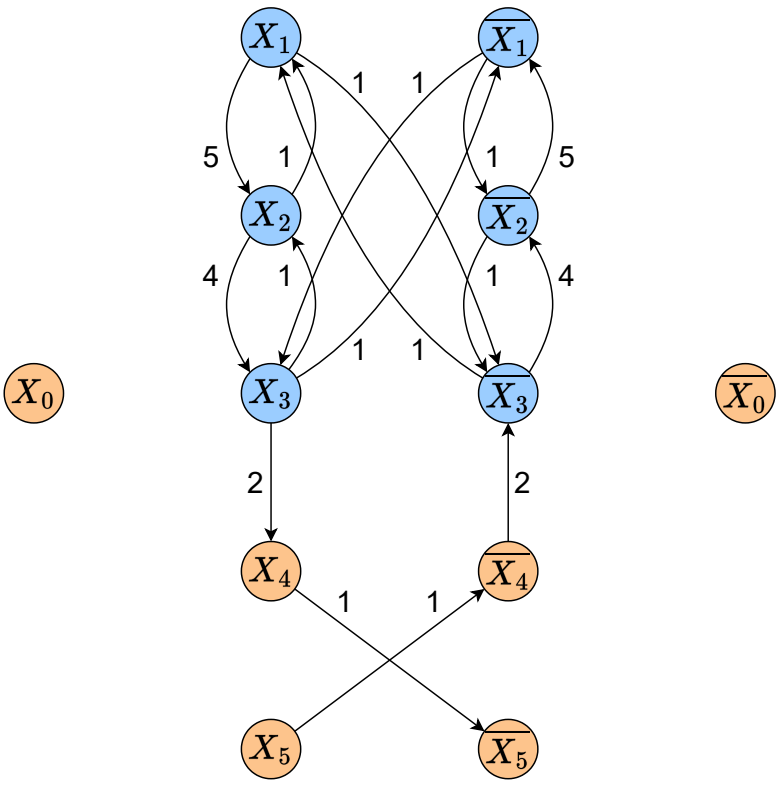


Figure 7.3: Residual network (CSCC highlighted in blue)

The nodes colored in blue $\{x_1, x_2, x_3, \overline{x_1}, \overline{x_2}, \overline{x_3}\}$ form a complete strongly connected component. If the optimization problem is solved without applying decomposition six different optimal solutions can be obtained:

x_1	x_2	x_3	x_4	x_5	$f(x)$
0	0	0	0	0	4
0	0	0	0	1	4
0	0	0	1	0	4
0	0	1	1	0	4
0	1	1	1	0	4
1	1	1	1	0	4

As explained in this section, the complete strongly connected component can be extracted from the network, transformed back to a function form employing the reverse of the process used to create the implication network, and solved independently. Then, the solution achieved can be substituted in the function associated with the network and it is possible to find the solution of the obtained function knowing its minimum in advance, that is 0 since it is a purely quadratic posiform. Therefore, extracting the CSCC and transforming the networks in posiform two functions are obtained:

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

For the function without CSCC, the variables x_1 and x_2 have been removed since they do not have any term in common with x_4 and x_5, therefore they do not affect the solution. If the posiform associated with the CSCC is solved, four different optimal solutions are obtained:

x_1	x_2	x_3	$\Psi_{CSCC}(x)$
0	0	0	4
0	0	1	4
0	1	1	4
1	1	1	4

At this point if the two possible assignments for x_3 are substituted into Ψ_{noCSCC}, two different functions are obtained:

- If $x_3 = 0$:

$$\Psi'_{noCSCC} = 2x_4x_5.$$

The optimal values for this posiform are:

x_4	x_5	Ψ'_{noCSCC}
0	0	0
0	1	0
1	0	0

- If $x_3 = 1$:

$$\Psi''_{noCSCC} = 4\overline{x_4} + 2x_4x_5.$$

The optimal values for this posiform are:

x_4	x_5	Ψ''_{noCSCC}
1	0	0

It is remarked that when the decomposed solutions are recombined together, they are exactly equivalent to the results attained without decomposition. △

7.3 Problem-specific decomposition

Problem-specific decomposition methods exploit the features of a certain class of problems to divide them into smaller ones or make assumptions and discard non-optimal solutions. In this section, it is described a decomposition routine for maximum clique (MC) and minimum vertex cover (MVC) problems [35].

In both cases, in order to split up the graph, the subject of the problem, into two smaller ones, a vertex v is identified to obtain two subgraphs G^+ and G^-, in which it is assumed that in G^+, v belongs to the optimal solution and in G^-, it does not. The best choice for v is the vertex with the lowest degree (i.e. number of connections of a vertex) for maximum clique and the one with the highest degree for minimum vertex cover. Successively, the splitting routine employed is problem dependent. At this point, it is possible to discard one of the subgraphs, if its solution can not be better than the best one found at the time being or an estimated one. For maximum clique, if the subgraph can have the best solution smaller than the current best one, it can be discarded. Alternatively, for minimum vertex cover, if the subgraph can have the best solution larger than the current best one, it can be discarded. Furthermore, vertex and edge removal techniques can be used to reduce the size of the graph. If a subgraph contains more vertices than a predefined cutoff value, it is possible to proceed recursively splitting it again into two subgraphs. A cutoff value can be defined as the maximum number of vertices for which it is possible to solve the graph with the available hardware. When this limit is reached the optimization problem in the leaf subgraph is solved and the best solution is updated for each recursion. Let μ be the value of the objective function, updated at each iteration, that is initialized as ∞ if the problem is a minimization or as $-\infty$ otherwise.

7.3.1 Maximum clique splitting routine

The maximum clique specific splitting routine generates the subgraph G^+ taking all the nodes connected by an edge with the vertex v and the subgraph G^- removing the vertex v and all the edges connected to it from the graph G. For G^+, $\mu^+ = \mu+1$ since the removed vertex v is part of the clique, while for G^-, $\mu^- = \mu$. In both cases, the graph size is reduced by at least 1, since v is removed. Therefore, the termination of the algorithm is guaranteed.

In order to reduce the number of nodes and edges of the graph for the max clique problem the vertex and node **k-core algorithms** can be employed. The k-core of a graph is the maximal subgraph in which each node has at least degree k. A clique of size k+1 is contained in the k-core, so, if a lower bound to use as value k is known, all the nodes outside the k-core can be removed. A lower bound can be obtained by applying the DDT heuristic disclosed in Section 6.3.1, or it can be defined by the subgraphs already solved. For instance, if a subgraph has a size of four and the solution of another previously processed subgraph is five, the current subgraph can be discarded. The edge k-core algorithm [36], first selects a random vertex v in the k-core subgraph. Then, all the edges (v, n) for which the following relation holds:

$$|N(v) \cap N(n)| < lowerbound - 2, \tag{7.9}$$

(where $|N(v) \cap N(n)|$ is the intersection of the neighbors, i.e., nodes connected by an edge, of v and n) are removed since they can not be part of a clique with a size higher than the current lower bound.

Example. *Maximum clique splitting routine.*
Take the graph in Figure 7.4 and apply the decomposition technique to find the maximum clique.

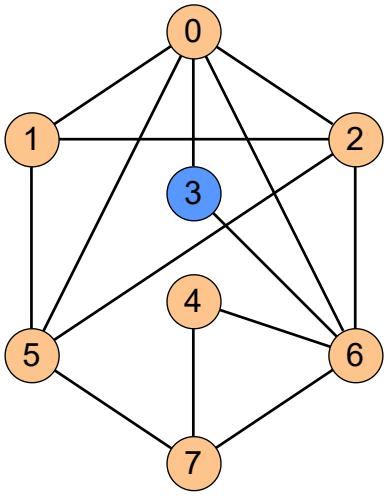
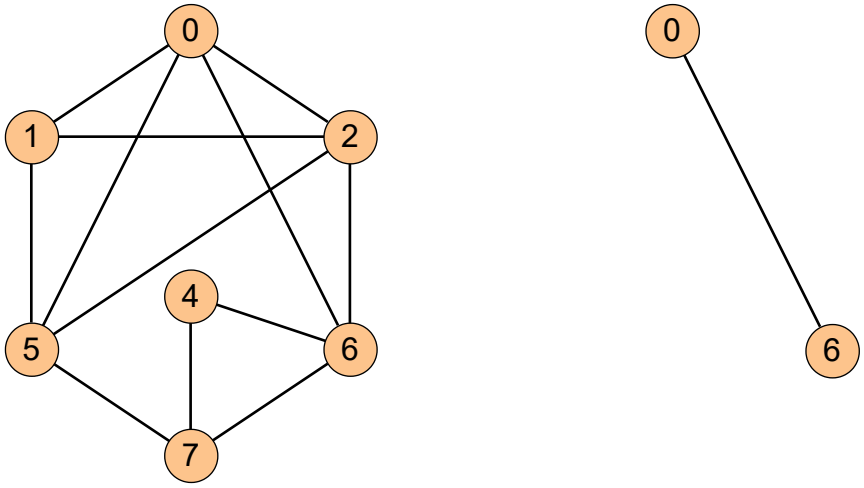


Figure 7.4: Example graph

The vertex selected (colored in blue in Figure 7.4) is vertex 3, since it is the one with the lowest degree. Now decomposition is applied following the rules described in this section and the subgraphs G^+ and G^- are obtained (Figure 7.5).



(a) G^- subgraph (b) G^+ subgraph

Figure 7.5: Subgraphs obtained by the decomposition

If the problem on graph G^- is solved first, a maximum clique with a size greater than 2 is attained, therefore the graph G^+ can be discarded since it can not provide a better solution. To furtherly reduce the number of vertices and edges in the graph G^- the k-core algorithms can be applied.

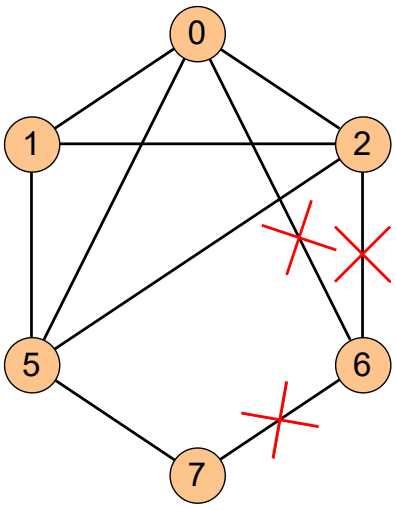


Figure 7.6: G^- after vertex and edge 3-core

Suppose, for example purposes, that a lower bound on the dimension of the

clique is known and it is equal to 4. Then, all the vertices part of the clique are also part of the 3-core of G^-. The only vertex with a degree lower than 3 is vertex 4. Hence it can not be part of the clique, since it has less than 3 edges, and can be removed. Finally, the edge k-core algorithm can be applied and suppose that vertex 6 is selected. It has 3 edges that connect it with 3 different vertices: 0 with which it has one neighbor in common (2) and 2 and 7 with which it has 0 neighbors in common. All the edges whose endpoints have less than 2 neighbors in common can be eliminated, thereby all of them are deleted (Figure 7.6). △

7.3.2 Minimum vertex cover splitting routine

The minimum vertex cover specific splitting routine generates the subgraph G^+ removing all the edges that have as endpoint the vertex v. This vertex is supposed to be part of the MVC, so all the edges connected to it are already covered, thus they are not part of the MVC and can be eliminated together with v. The subgraph G^- is generated considering that all the u vertices connected to v are part of the MVC because the edges (u, v) between them and v must be covered. For this reason, they can be removed (updating μ), and also all edges that have as endpoint a vertex u can be discarded because they are already covered. μ is updated according to the nodes added to the MVC in each recursion.

Example. *Minimum vertex cover splitting routine.*
Take the same graph used for the example for maximum clique (Section 7.3.1).

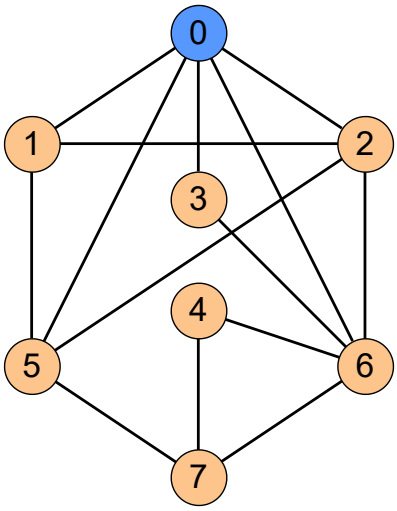
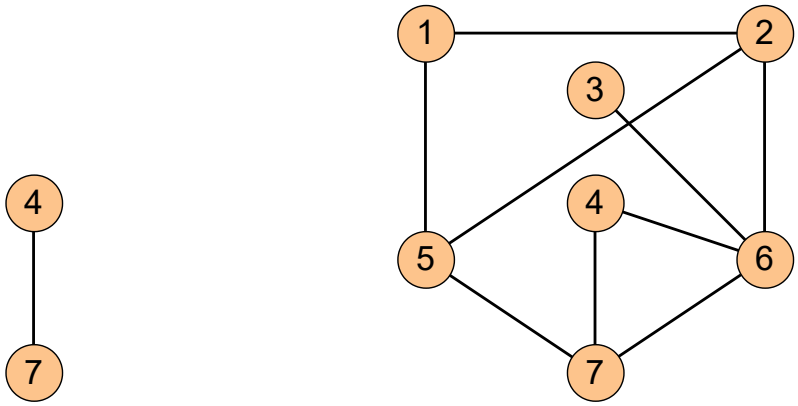


Figure 7.7: Graph with selected vertex highlighted

The best choice on the vertex to be used for applying the decomposition is the

one with the highest degree, that is vertex 0 in this case (highlighted in Figure 7.7). Following the rules described in this section two subgraphs G^+ and G^- can be obtained and they are shown in Figure 7.8.



(a) G^- subgraph (b) G^+ subgraph

Figure 7.8: Subgraphs obtained by the decomposition

For instance, if G^+ is solved first, it can be found that the solution has size 5. To construct G^-, all the 5 nodes connected to vertex 0 are supposed to be part of the cover, thus it has a size larger than the best solution and it can be discarded. The routine can be applied recursively until the subgraphs have a size smaller than a predefined cutoff. △

7.4 Shannon decomposition

The Shannon decomposition method name is derived from the **Shannon expansion theorem** for boolean algebra, since it is exploited to decompose a QUBO cost function. The theorem states that for a boolean function F and a variable x_i:

$$F = x_i F_{x_i} + \overline{x_i} F_{\overline{x_i}}, \tag{7.10}$$

where F_{x_i} and $F_{\overline{x_i}}$ are respectively the function F evaluated for $x_i = 1$ and $x_i = 0$. Since variables are binary, this also holds for a pseudo-boolean function, or in particular, for a QUBO one. If $x_i = 1$, $f = f_{x_i}$, whereas if $x_i = 0$, $f = f_{\overline{x_i}}$, thus this relation is still valid. Therefore, by applying the Shannon expansion theorem, it is possible to decompose a cost function in two subfunctions with one less variable. This method is always applicable, but, if executed iteratively for all the n variables belonging to the function f, 2^n evaluations of f are needed, hence it is equivalent to a brute-force approach that tries all the possible input configurations to find the minimum value. Clearly, this is not an efficient way to

try to solve an NP problem. Anyway, the Shannon decomposition can be employed after all the variable reduction and decomposition techniques treated in the previous sections to ultimately prune the QUBO function, if the number of variables is still too high for the qubits availability of quantum hardware. If just k variables have to be removed, the QUBO function can be decomposed in 2^k subfunctions, using k recursion levels. Then, after finding the minimum value of all of them, the optimal solution is given by the one obtained from the subfunction with the lowest minimum value merged with all the expansion variables assignments set to attain that particular subfunction.

Furthermore, consider a vertex v belonging to a strongly connected subgraph. v is a **strong articulation point** [37], if its removal makes the subgraph not strongly connected. When removing a strong articulation point, either the subgraph becomes made up of two strongly connected components or there are no strongly connected components at all. This implies that, if the expansion variable chosen for the Shannon decomposition corresponds to an articulation point in the residual network associated with the cost function, it can break a complete strongly connected component. Therefore, if two complete strongly connected components are formed, the strong components decomposition (Section 7.2) can be used to decompose the network furtherly. Otherwise, if no strongly connected components are present all the persistency-finding techniques can be applied, thus eliminating more variables from the QUBO function.

Example. *Breaking a strongly connected component with Shannon decomposition.* Consider the minimization problem of the following cost function:

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

After computing, the corresponding posiform, constructing the implication network and calculating the maximum flow, the residual network in Figure 7.9 is obtained. It is easy to observe that the residual network is composed of a unique complete strongly connected component that includes all the variables. In such a network, no persistency can be found and no general decomposition technique can be applied. Now suppose to apply Shannon decomposition and select as the expansion variable the one associated with the node with the highest degree. Namely, the variable that appears in most terms of the function. In this way, the largest amount of terms, or edges, is removed after the function evaluation. Therefore, the variable x_2 is chosen. Then, the two subfunctions are obtained evaluating f for $x_2 = 0$ and $x_2 = 1$:

$$f_{\overline{x_2}} = 12 - 4x_3 - 4x_1x_4 + 4x_3x_4.$$
$$f_{x_2} = 12 + 4x_1 - 4x_4 - 4x_1x_4 + 4x_3x_4.$$

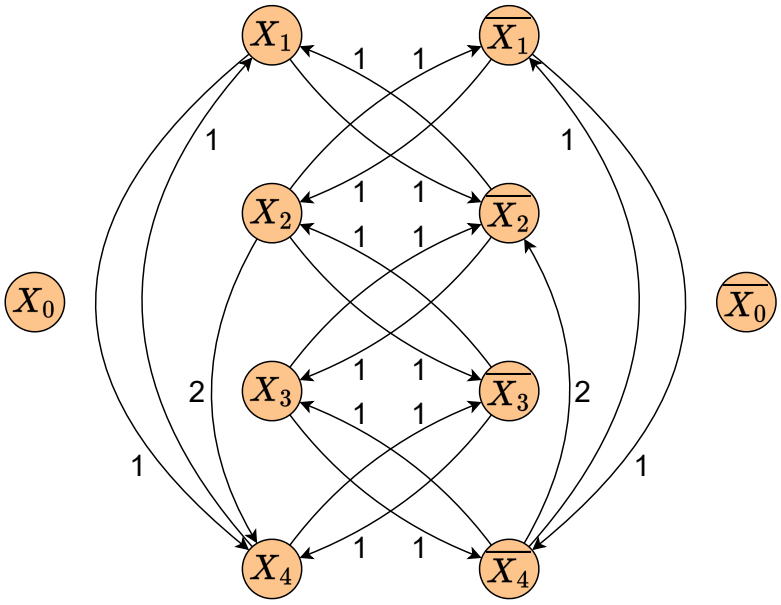


Figure 7.9: Residual network with unique complete strongly connected component

The two subfunctions produce two different implication networks. The network corresponding to $f_{\overline{x_2}}$ can be transformed into a residual network by calculating the maximum flow. The result is shown in Figure 7.10. In the network corresponding to f_{x_2}, the maximum flow is 0 because there is no path from source to sink, hence the network is already in the form of a residual one. It is reported in Figure 7.11.

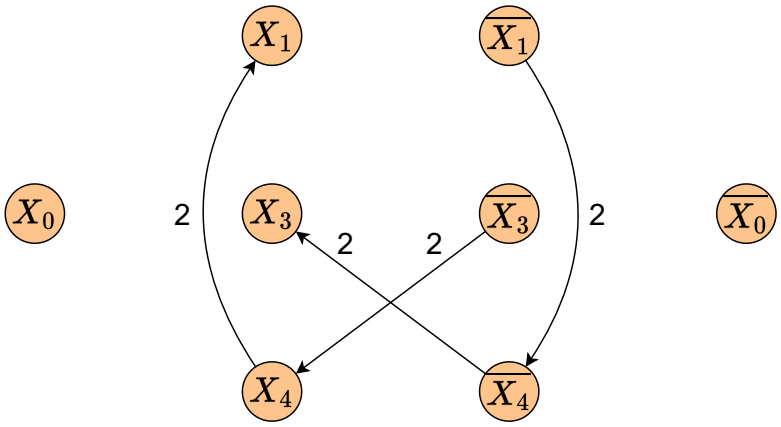


Figure 7.10: Residual network associated with $f_{\overline{x_2}}$

In the network in Figure 7.10, no persistency-finding technique and no general decomposition method can simplify the network. However, from a graph only composed of a complete strongly connected component a graph with none of them has been attained. Therefore the minimum value of $f_{\overline{x_2}}$ is known and it is equal to

8. The minimum of a CSCC-free purely quadratic posiform is 0 and by adding the constant term (4) and the maximum flow value (4) the correct result is achieved. For this reason, as proven in Section 7.2, the optimal input configuration of $f_{\overline{x_2}}$ can be found just by exploiting Grover's search instead of Grover Adaptive Search, achieving a considerable speedup.

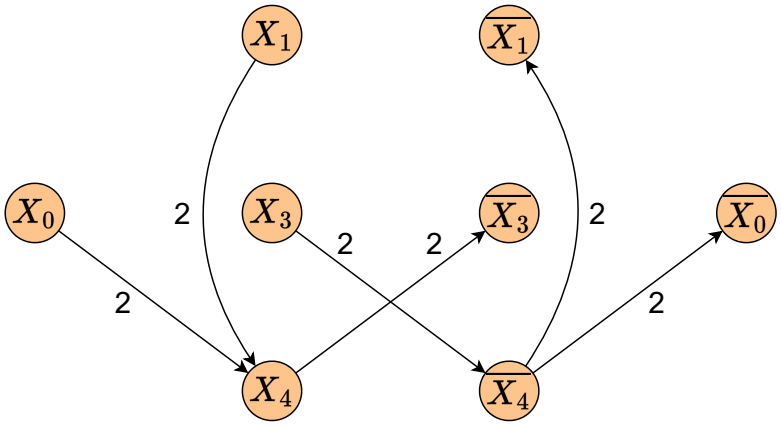


Figure 7.11: Residual network associated with f_{x_2}

In the network in Figure 7.11, $\overline{x_3}$ and x_4 can be reached from the source, therefore they are strong persistencies (as explained in Section 6.1). x_1 remains alone in the network, thus it can be equal to both binary values. From a complete strongly connected component in which no variable assignments can be known in advance, thanks to Shannon decomposition, if $x_2 = 0$, the optimal values of all the other variables are known. they are $x_1 = 0{,}1$, $x_3 = 0$, $x_4 = 1$. △

Part III

Conclusions

Chapter 8

Results

In this chapter, all the results obtained by executing the toolchain on different benchmark problems, of different dimensions, are reported and explained in detail. Python scripts have been used to randomly generate several instances of each problem, with a certain span of the number of variables and density. These scripts, then, call the toolchain functions and the solver interfaces provided by external libraries. They are all executed on a server, whose characteristics are described in [38].

In Section 8.1, the minimum vertex cover (described in Section 2.4.1) is used as a benchmark to observe the number of persistencies found, decompositions applied and the total decrease of the size of the problem, as functions of the characteristics of the QUBO matrix. In Section 8.2 and Section 8.3, the same analysis is carried out on two other benchmarks, that are respectively maximum clique (Section 2.4.2) and maximum cut (Section 2.4.3). In Section 8.4, the toolchain has been used to enhance the capabilities of the GAS solver. It is shown that the toolchain can be used to solve problems with larger sets of variables and it enables achieving a faster convergence to the solution.

8.1 Minimum vertex cover

The minimum vertex cover problem is used as a benchmark to verify the effectiveness of the methods employed in the toolchain. The goal is to find the fraction of persistencies identified, the number of decompositions applied and the accuracy of the lower bound estimation as functions of the size of the problems, which can be expressed as the number of variables in the corresponding QUBO formulation, and the density of the QUBO matrix. The density of the graph is defined as the probability of having an edge between a couple of nodes, or, similarly, the fraction of existing edges on the combination of all the possible couples of nodes. Moreover, expressing the QUBO formulation in matrix form, its density can be defined as the probability of a non-diagonal term being different from 0, i.e. of having a quadratic

coefficient different from 0. Since in the minimum vertex cover QUBO formulation there is a quadratic term anytime there is an edge between two nodes, the density of the QUBO matrix is directly proportional to the density of the graph.

The problem instances are produced by constructing graphs with the desired number of nodes, where an edge between two nodes is generated according to a probability equal to the desired density. One thousand and forty graphs have been created with the following characteristics:

- number of nodes ranging from a minimum of 10 nodes to a maximum of 70 with a step of 2;
- four different density values: 2%, 5%, 10%, 20%;
- ten different randomly generated edge sets for each size and density.

Then, each problem is solved by providing its corresponding QUBO matrix as input to the toolchain, which gives as output persistencies and simplified and decomposed QUBO functions, which are finally solved using the simulated annealing algorithm exploiting the qubovert Python library [17]. To verify their correctness, the results obtained are compared with the solution attained by directly solving each problem with the simulated annealing. Since the simulated annealing is a heuristic algorithm, it may find a suboptimal solution, hence an error is eligible, but it is expected to be very low with respect to the range of values assumed by the cost function. For every problem, the error has been computed as the difference between the minimum obtained with the use of the toolchain and the minimum obtained just with simulated annealing. An average error of 0.1 and a maximum of 3 with respect to cost function values up to around 4000 have been obtained, which have been considered sufficient to prove the validity of the results.

Figure 8.1 shows the average percentage of variables that can be identified as persistencies. For the lowest density, the percentage is almost always 100%. Therefore, the toolchain is very efficient in this case, allowing to solve problems without even resorting to any solver. However, the higher the density and the number of nodes, the lower the probability to find a high percentage of persistencies. Indeed, a higher number of terms in the QUBO formulation leads to a denser implication network, i.e. with a larger amount of edges, thus resulting in greater difficulty in finding relations between variables.

Figure 8.2 displays the average number of trivial decompositions. It can be noticed that the occurrences of residual networks made up of disconnected components are very rare. This is because for lower densities, when it is more probable for the network to have isolated sets of nodes, the persistencies-finding techniques are able to identify the optimal assignments for them. Hence by removing these variables, the disconnected components disappear. For high densities, instead, there is just a lower probability to have disconnected components in the first place.

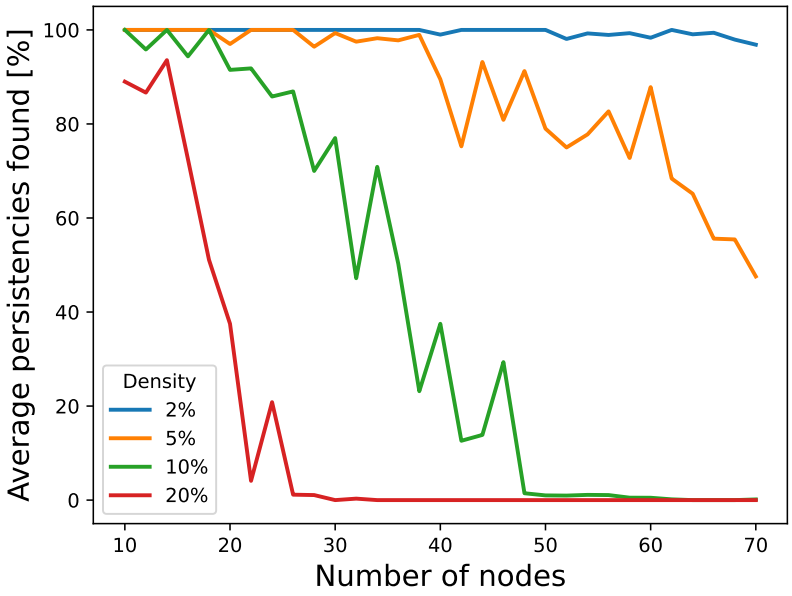


Figure 8.1: Average percentage of persistencies found

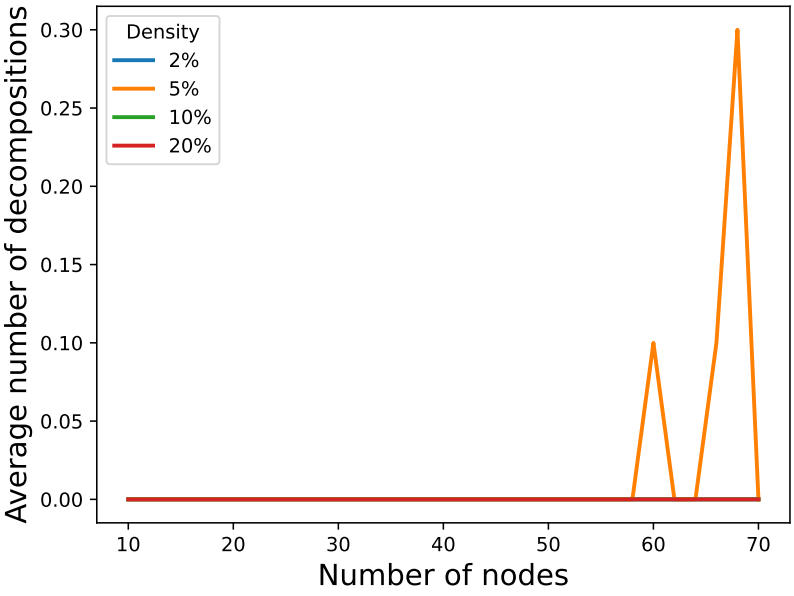


Figure 8.2: Average number of trivial decompositions

Figure 8.3 reports the average number of CSCC decompositions, i.e. the number of times that a CSCC has been identified in the residual network and extracted. The number of CSCCs increases with the number of nodes and the density of the QUBO matrix, and consequently, of the residual network. This trend is due to

the fact that a higher density means a higher number of edges, thus it is more probable to have paths that can reach all the other variables. With the same density, a greater number of nodes also increases this probability. It is noteworthy that the profiles of the number of CSCCs decompositions and the persistencies found are approximately one the opposite of the other. This similarity tells that the persistency-finding techniques are particularly effective when no CSCCs are present. Then, increasing the density, the probability to have a CSCC increases and consequently the probability of finding persistencies decreases. On one hand, this further validates the current techniques for low density and on the other hand, it sets the goal for the next methods to be added, namely to find persistencies or apply decomposition that allow breaking and reducing CSCCs. It should be also observed that it is extremely rare, and in this case never occurs, that two or more independent CSCCs are present in the network. Hence, CSCCs for large problems with high densities are an obstacle to finding persistencies and have a dimension that is still big enough to not enable the use of GAS in current quantum backends.

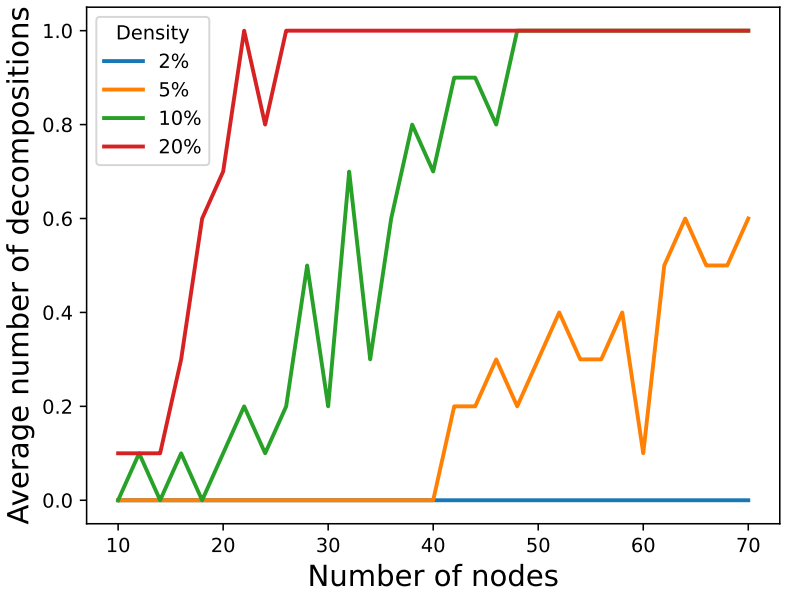


Figure 8.3: Average number of CSCC decompositions

Figure 8.4 shows the total reduction of the number of variables, i.e. the difference, as a percentage computed with respect to the dimension of the original problem, between the size of the problem itself and that of the biggest subproblem provided by the toolchain. The curves in this plot are very similar to the persistencies ones, therefore the main contribution to the reduction of the number of variables is given by persistencies. This enforces the observation that persistency-finding techniques are effective when CSCCs are not present. Instead, when they

are, their number is very often equal to 1, so decompositions do not have a strong effect on the dimension of the problem. The differences between these two graphs are due to CSCC decomposition when the residual network, after persistencies have been removed, is composed of a CSCC and another subset of nodes, so that when these two components are divided the number of nodes is decreased.

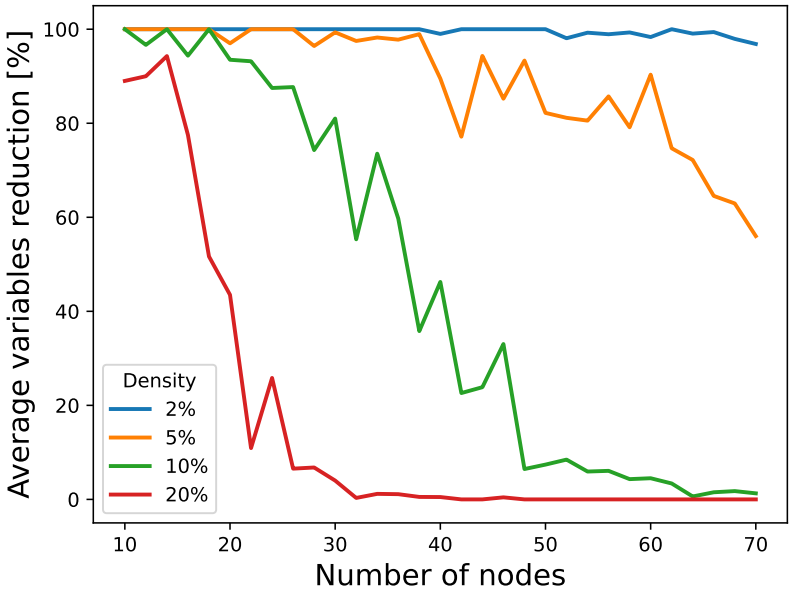


Figure 8.4: Average percentage of the total reduction in the number of variables

Figure 8.5 reports the error on the lower bound estimation, expressed as the difference between the minimum value of the cost function and the lower bound obtained. Also in this case, if the density and the number of nodes increase the lower bound gets worse, since more edges are present and more iterations of Probing would be needed to further increase the maximum flow in the network. If an upper bound is symmetrically computed, it would enable the estimation of the number of qubits necessary to encode the values in the quantum dictionary. Supposing to have an equal error on the upper bound, the values found are very accurate and would result in an error on the value qubits required most of the times of no more than one. The number of qubits estimated is correct if the span of values assumed by the function, considering also the shifts performed by the GAS algorithm, can be encoded by the same number of qubits also including the error. For example, with an error of 20 for both bounds, the span must be lower than 16 to have a difference of more than two qubits between the ideal number and the estimated one. If the span is larger than 16 at least 5 qubits are necessary to encode the values and $2^5 + 20 < 2^6$, hence only one more qubit is required. However, when the error on the lower bound estimation is 20 the number of nodes is 70, so a typical span of

values assumed by the function is 3500 which is way greater than 32 and this is not even considering the shifts that GAS performs that would make the actual span of values larger. Of course, an error of 1 qubit should always be considered because if n qubits are ideally needed to encode the values of a cost function with a span of $2^n - 1$, an error of 1 in the lower bound estimation would result in an extra qubit.

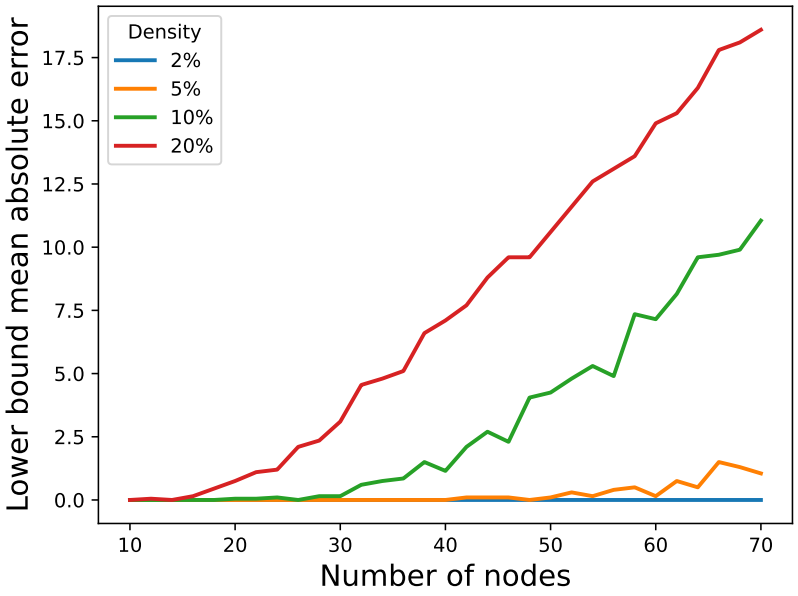


Figure 8.5: Average error on lower bound estimation

8.2 Maximum clique

The maximum clique problem has been employed as another benchmark to check the correctness of the results obtained with the use of the toolchain and the effectiveness of the methods to reduce the size of the problems. In the maximum clique QUBO formulation (see Section 2.4.2), it is added a quadratic term every time there is no edge between a couple of nodes. Therefore, the density of the graph in which the clique has to be found is inversely proportional to the density of the corresponding QUBO matrix. The graphs have been constructed by inserting an edge between two nodes according to a probability equal to the desired density. One thousand and forty graphs have been created with the following characteristics:

- number of nodes ranging from a minimum of 10 nodes to a maximum of 70 with a step of 2;
- four different density values: 80%, 90%, 95%, 98%;
- ten different randomly generated edge sets for each size and density.

As described in Section 8.1, the simulated annealing algorithm has been used to solve the problems provided as outputs by the toolchain and to verify the correctness of the results obtained. An error equal to 0 has been attained for all the problem instances, therefore the solutions are all to be considered valid.

The results obtained for this benchmark are very similar to the minimum vertex cover ones. The densities indicated in the plots are referred to graph subject of the problem, i.e. where the clique has to be found. Therefore, the highest density corresponds to the lowest density of the QUBO matrix. Figure 8.6 shows the average fraction of persistencies detected. Also in this case, as the density and the number of nodes increase the number of persistencies found decreases and the techniques are very efficient for low densities.

Trivial decompositions are very rare since persistencies make the isolated terms disappear. Therefore, it is possible to derive the conclusion that the presence of disconnected components after the elimination of the persistencies from the residual network is mainly random or very uncommon for the characteristics of the graphs produced. In this particular test, they never occur, for this reason, their plot is not reported.

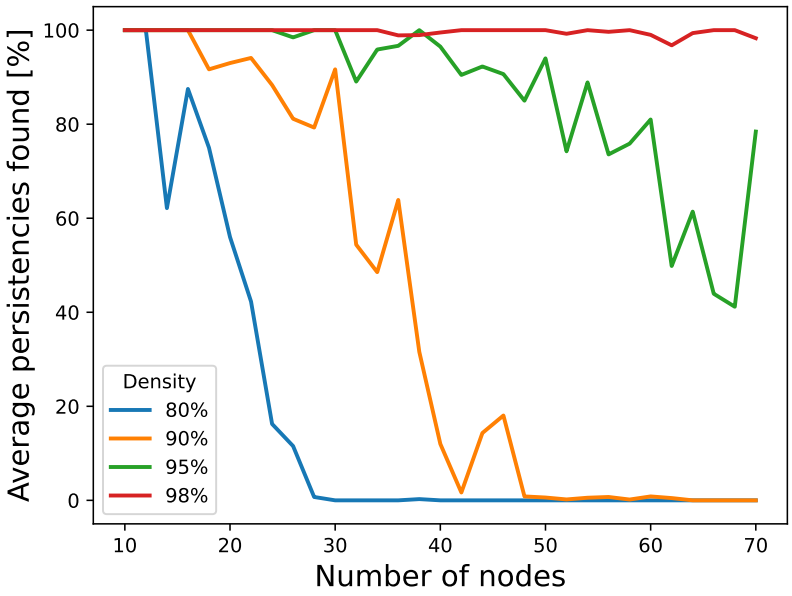


Figure 8.6: Average percentage of persistencies found

CSCC decompositions (Figure 8.7) follow the opposite profile of the persistencies, therefore the persistency-finding techniques are very effective when there are no CSCCs, and also for maximum clique the issue to be solved to obtain problems of reduced size is to break or decrease the dimension of CSCCs.

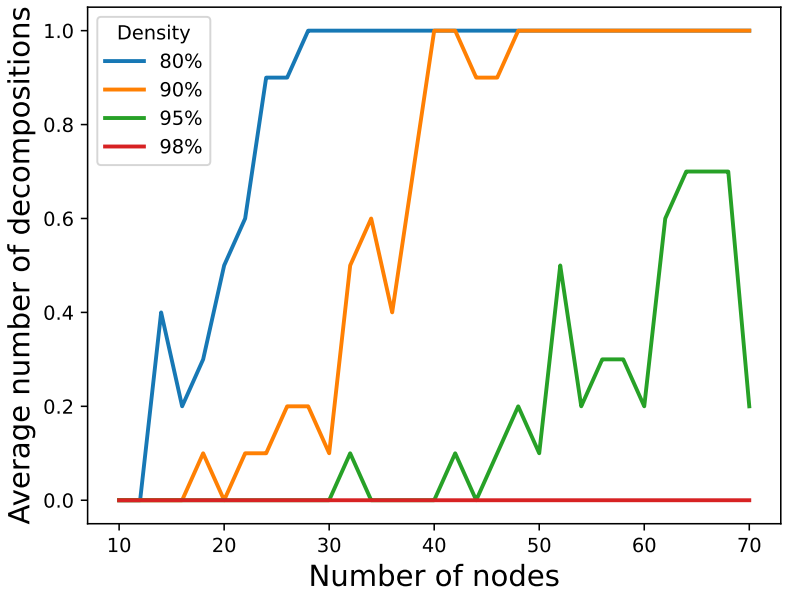


Figure 8.7: Average number of CSCC decompositions

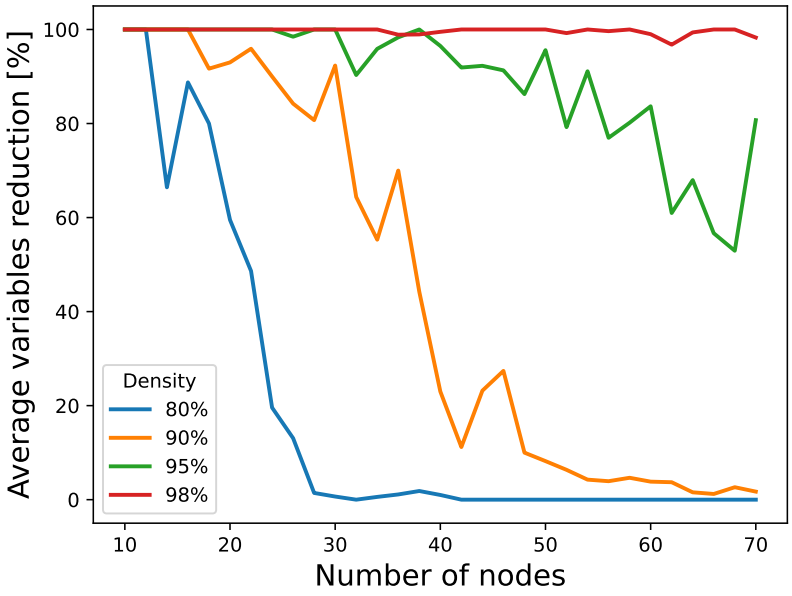


Figure 8.8: Average percentage of the total reduction in the number of variables

The total reduction of the number of variables (Figure 8.8) is, similarly to the minimum vertex cover case, dominated by the removal of persistency. Especially for high density and a high number of nodes, it can be seen the effect of CSCC

decomposition which causes an additional reduction of variables to what was already obtained by the persistencies. For the 95% density curve between 60 and 70 nodes, it can be observed a further reduction of the size of the problem of around 10%. This effect is also evident in the 90% density curve for problems larger than 40 variables.

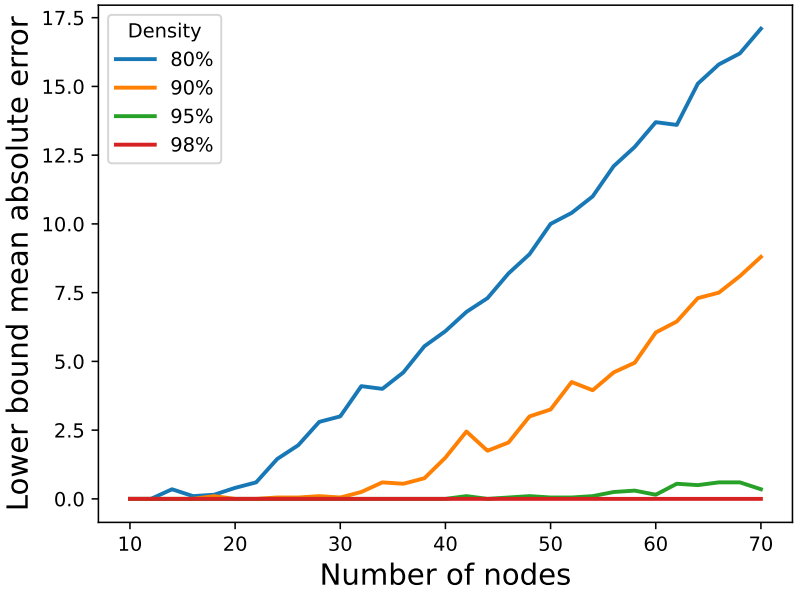


Figure 8.9: Average error on lower bound estimation

The error on the lower bound estimation (Figure 8.9) follows the same trend as the minimum vertex cover case, but its maximum is even smaller, arriving at 17 for problems with 70 variables.

8.3 Max-cut

The max-cut problem has been used as a benchmark to check the correctness of the results obtained with the use of the toolchain and the effectiveness of the methods to reduce the size of the problems. In the max-cut QUBO formulation (see Section 2.4.3), it is added a quadratic term and two linear terms every time there is an edge between a couple of nodes. Therefore, the density of the graph in which the cut has to be found is directly proportional to the density of the corresponding QUBO matrix. The graphs have been constructed by inserting an edge between two nodes according to a probability equal to the desired density. Unlike the minimum vertex cover and maximum clique cases, the element to minimize or maximize is not the number of nodes but the number of edges, hence a variable appears in a linear or a quadratic term only if it has at least one edge. For this reason, it is

not possible to construct a graph that leads to a QUBO matrix with a predefined number of variables and a low density. Otherwise, it would be possible to have a node that does not appear in the QUBO formulation, thus resulting in a number of nodes, and consequently, a density, different from the predefined ones. Therefore, for this particular benchmark, 9% has been chosen as the lowest density value. One thousand and forty graphs have been created with the following characteristics:

- number of nodes ranging from a minimum of 10 nodes to a maximum of 70 with a step of 2;
- four different density values: 9%, 12%, 15%, 20%;
- ten different edge sets for each size and each density;
- weight of an edge, when present, randomly assigned to an integer number between 1 and 4.

As described in Section 8.1, the simulated annealing algorithm has been used to solve the problems provided as outputs by the toolchain and to verify the correctness of the results obtained. An error equal to 0 has been attained for all the problem instances, therefore the solutions are all to be considered valid.

The results for this benchmark differ from the other ones mainly because cliques and vertex covers depend only on the presence or absence of edges, whereas the maximum cut also depends on the weight of the edges.

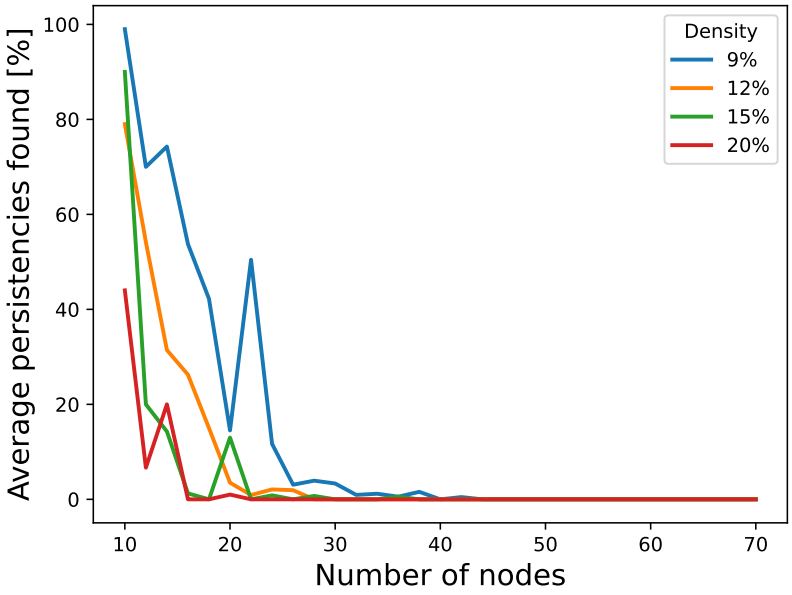


Figure 8.10: Average percentage of persistencies found

The trends of the plots are the same as the previous benchmarks, but the results are slightly worse. The average persistencies percentage (Figure 8.10) decreases faster as the density of the graphs subject of the problem, and consequently of the QUBO matrices, increase.

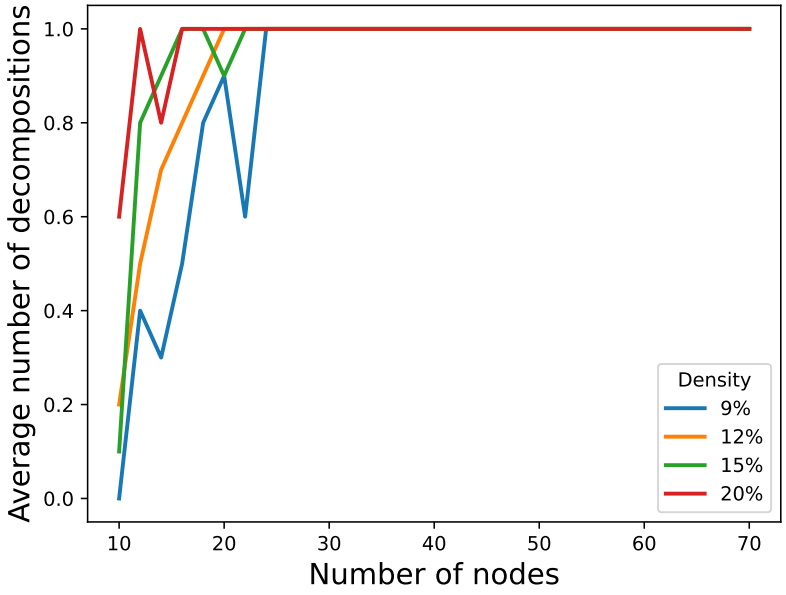


Figure 8.11: Average number of CSCC decompositions

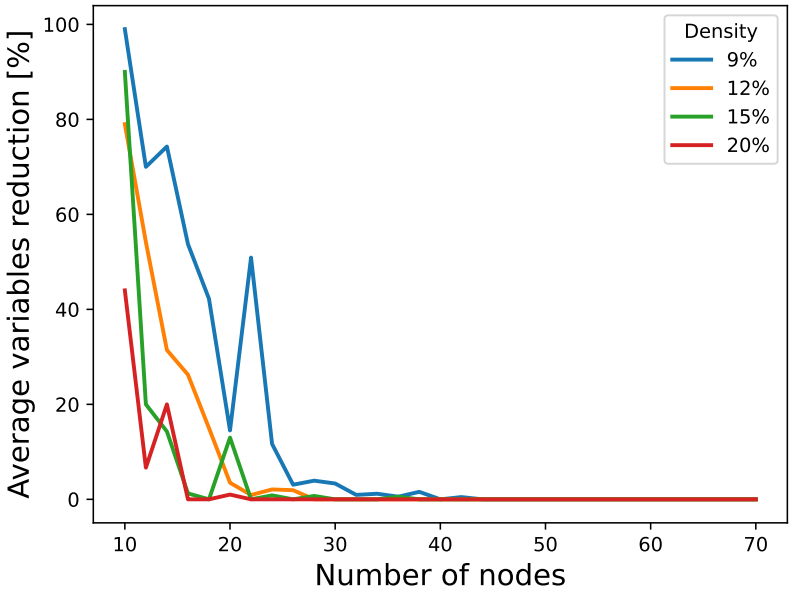


Figure 8.12: Average percentage of the total reduction in the number of variables

The trivial decompositions are not reported since the occurrences of disconnected graphs are mainly random and extremely few. The average number of CSCC decomposition (Figure 8.11), in accordance with the persistencies plot, has a faster increase as the density of the graphs increases, thus maintaining the complementarity with it. This behavior can be explained by the fact that there are different edge weights in the graphs which result in a greater heterogeneity of the coefficients in the QUBO matrix. Hence, when calculating the maximum flow, it is more probable that for an edge the flow is lower than its weight which leads to the existence of two edges with opposite directions that increase the probability of having strongly connected components. As a consequence, since all the results prove this correlation, if the probability of finding a complete strongly connected component increases, the probability of finding persistencies decreases.

The total reduction in the number of variables (Figure 8.12) follows the previous trends as well, but it is obviously influenced by the lower amount of persistencies found.

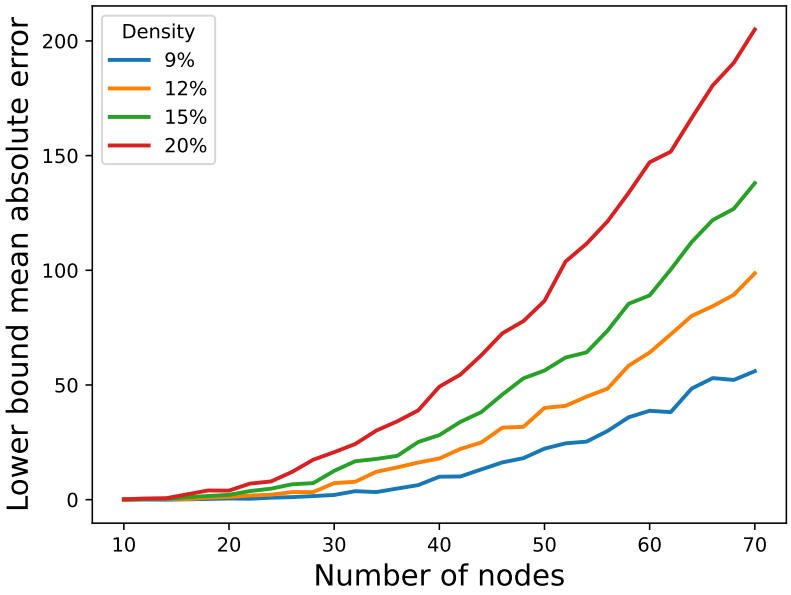


Figure 8.13: Average error on lower bound estimation

The error on the lower bound estimation (Figure 8.13) is considerably higher than the previous cases. This can be explained by the fact that if CSCCs with higher degrees of connectivity are present, the minimum of the cost function is increased with respect to a CSCC-free case because of the presence of many quadratic terms that is not possible to make vanish. Thus, the initial lower bound, calculated with the initial maximum flow, is further from the real minimum. Several recursions of Probing would be needed to better estimate this bound. However, this result can still be considered sufficient to guarantee an accurate estimation of the number of

qubits necessary to encode values in the quantum dictionary.

8.4 Exploitability of GAS simulator

The ultimate goal of the toolchain is to improve the chances of employing the GAS algorithm to solve real-world combinatorial optimization problems. Therefore, the minimum vertex cover problem has been used as a benchmark to compare the time taken to converge to the solution by the GAS solver after the execution of the preprocessing toolchain and the GAS solver alone. For this purpose, the execution time of the two different procedures has been chosen as figure of merit. The Grover search employed by the GAS algorithm is executed on the qasm simulator provided by Qiskit [19]. The problems are generated by producing random graphs in which the minimum vertex cover has to be found and they are then translated in QUBO formulation and given as input to the toolchain and the GAS solver. Two hundred graphs with the following characteristics have been created:

- two different density (defined as described in Section 8.1) values: 20%, 40%;
- number of nodes ranging from 3 up to 7 for problems to be solved only with GAS, up to 9 for problems preprocessed by the toolchain with 40% density and up to 13 for problems preprocessed by the toolchain with 20% density;
- ten different randomly generated edge sets for each size and density.

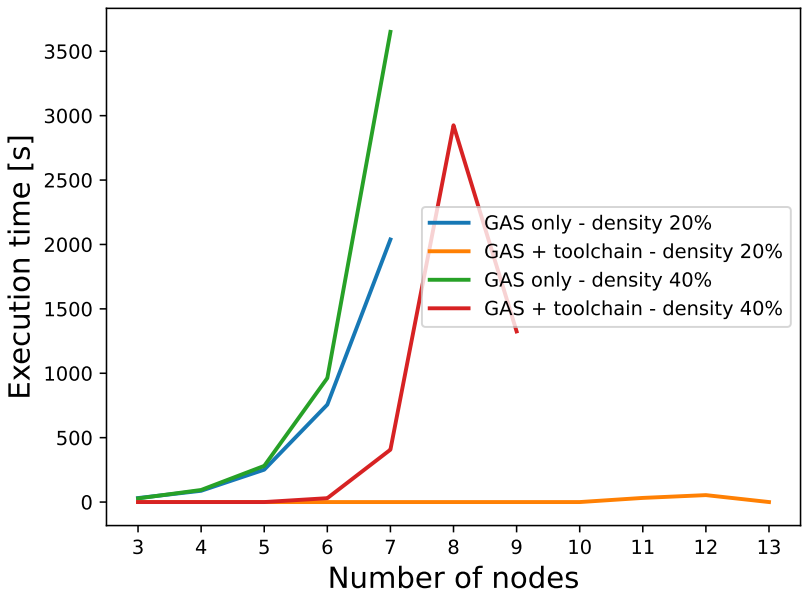


Figure 8.14: Comparison between the execution time of problems solved by running the toolchain before the GAS simulator or with GAS only.

The results obtained (Figures 8.14 and 8.15) show that the toolchain effectively improves the exploitability of the GAS algorithm and reduces the execution time. It can be observed that the solution of problems with 40% density and 8 or 9 nodes preprocessed with the toolchain takes a time of the same order of magnitude as problems with 7 nodes solved only by the GAS algorithm. Instead, for problems with 20% density the execution time up to 13 nodes is substantially lower and does not even scale exponentially. Therefore, preprocessing enables dealing with larger problems with respect to what is currently possible, since the GAS solver employs an unfeasible amount of time for problems with more than 8 variables and the limit of qubits that can be simulated is also overcome.

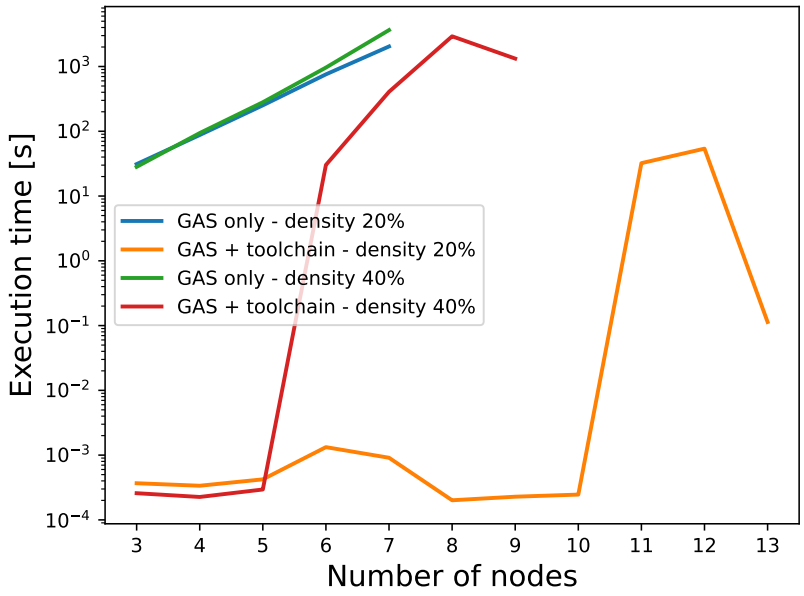


Figure 8.15: Comparison between the execution times with or without the use of the toolchain in logarithmic scale.

The efficiency of the toolchain and its ability to decrease execution time depends on the density of the problem at hand. Consequently, the efficacy of the toolchain for problems involving heavily interconnected graphs is limited. On the other hand, encouraging results can be obtained for problems like the minimum vertex cover, for which it is common to have a sparse matrix.

Chapter 9

Future perspectives

The preprocessing toolchain presented in this thesis is the first milestone that lays the foundations for a complete automation of quantum optimization processes, starting from the generation of the QUBO formulation of a combinatorial optimization problem and ending with the solution of the problem itself exploiting the available quantum hardware, together with the choice of the most suitable quantum algorithm to solve it.

First of all, a combinatorial optimization problem can be written as just a set of constraints, or in the form of a function, not necessarily in a QUBO form. Therefore, to implement the first two steps of a toolchain for quantum optimization, approaches like the one proposed in [39] can be employed. In addition, in Section 2.2, it has been described that penalty coefficients for embedding constraints in the QUBO formulation must be chosen within a certain range to guarantee that optimal solutions do not violate constraints and that it is easy to understand whether a solution is better than another one. Too high penalty coefficients can also enlarge the span of values assumed by the cost function leading to a greater need for qubits to encode these values in a quantum dictionary. Hence, together with an automatic generation of the QUBO formulation, a method for the estimation of penalty coefficients, as proposed in [40], can be added. Then, in order to accept any constraint expressed in inequality form, methods for the definition of the lowest possible number of slack variables, as presented in [41], have to be added, so that to keep the size of the problems small enough to be able to use quantum solvers. All the previous steps combined permit having a complete mechanism for QUBO recasting of the most general set of combinatorial optimization problems.

The toolchain currently works only with integer QUBO coefficients, therefore it can be improved by supporting floating points coefficients. For this purpose, a normalization step can be added, that adapts all the coefficients according to the value of the smallest one, recasting them to integers. Normalization can also be used to reduce the value qubits in the GAS algorithm, dividing all the coefficients by their greatest common divisor. In addition, knowing the range of values assumed

by the function, it is possible to apply an offset in order to start the GAS in a condition where few negative values are present, but it is ensured that there is at least one. In this way, the classical iterations needed to find the optimum value may be decreased.

Furthermore, dynamic policies explored in [20] to select the number of Grover rotations and a threshold for the stop condition in the Grover Adaptive Search algorithm can be employed to achieve faster convergence of the algorithm and a greater probability of success. It could be also possible to analyze which policy suits best a certain problem by observing the characteristics of its corresponding QUBO matrix, such as density or mean and variance of the coefficients.

The results show that, for what concerns variables reduction mechanisms, the main issue to be solved is the presence of large complete strongly connected components (CSCCs) in the residual networks, therefore the toolchain should be expanded with a method to break them or find valid assignments for variables belonging to these components. The Shannon decomposition is theoretically capable of breaking a CSCC or splitting it into two, but the outcome of the decomposition depends on the choice of the expansion variable. This election can be handled by an algorithm that can identify a strong articulation point, as demonstrated by [37]. However, the previous approach is able to break a strongly connected subgraph only if eliminating just one node is sufficient. Hence, to decompose any CSCC the best expansion variable selection algorithm should find the smallest subset of nodes whose removal makes a CSCC no longer strongly connected. In this way, a number of Shannon decomposition iterations equal to the number of nodes in the subset could split up any CSCC. Moreover, the current implementation of Probing in the toolchain does not take into account quadratic persistencies that could be included in the future to find new variable assignments in CSCCs. More quadratic persistencies could also be found by employing the methods presented in [42].

An automated quantum toolchain for combinatorial optimization problems (Figure 9.1), in general, would be necessary to provide an abstraction layer allowing non-experts to use the power of quantum computation for these purposes [43, 44]. This is, by the way, a long-term prospect since it would require embedding many different quantum solvers employing different quantum algorithms to choose the best fit for the problem at hand. The first step could be the automatic generation of the QUBO formulation given a user-friendly interface as expressed above. At this point, the preprocessing toolchain, subject of this thesis, with the described improvements, could be utilized to reduce and decompose the cost function and to provide the lower bound estimation, for instance, to determine the number of qubits necessary to encode the function for the GAS algorithm. Then, according to the characteristics of the QUBO formulation of each particular problem, such as the number of variables, the density of the matrix or mean and variance of the coefficients, the best quantum device or quantum paradigm could be chosen to achieve the desired solution with the best accuracy and the lowest execution

time. Each solver is preceded by a step for the selection of the best parameters for the problem at hand, for instance, the number of quantum rotations and threshold for the stop condition for the Grover Adaptive Search. Many solvers such as the quantum annealer, Quantum Approximate Optimization Algorithm (QAOA), Variational Quantum Eigensolver (VQE) and Grover Adaptive Search could be supported, but they are just a glimpse of all the algorithms and paradigms available, therefore this list can be expanded in the future.

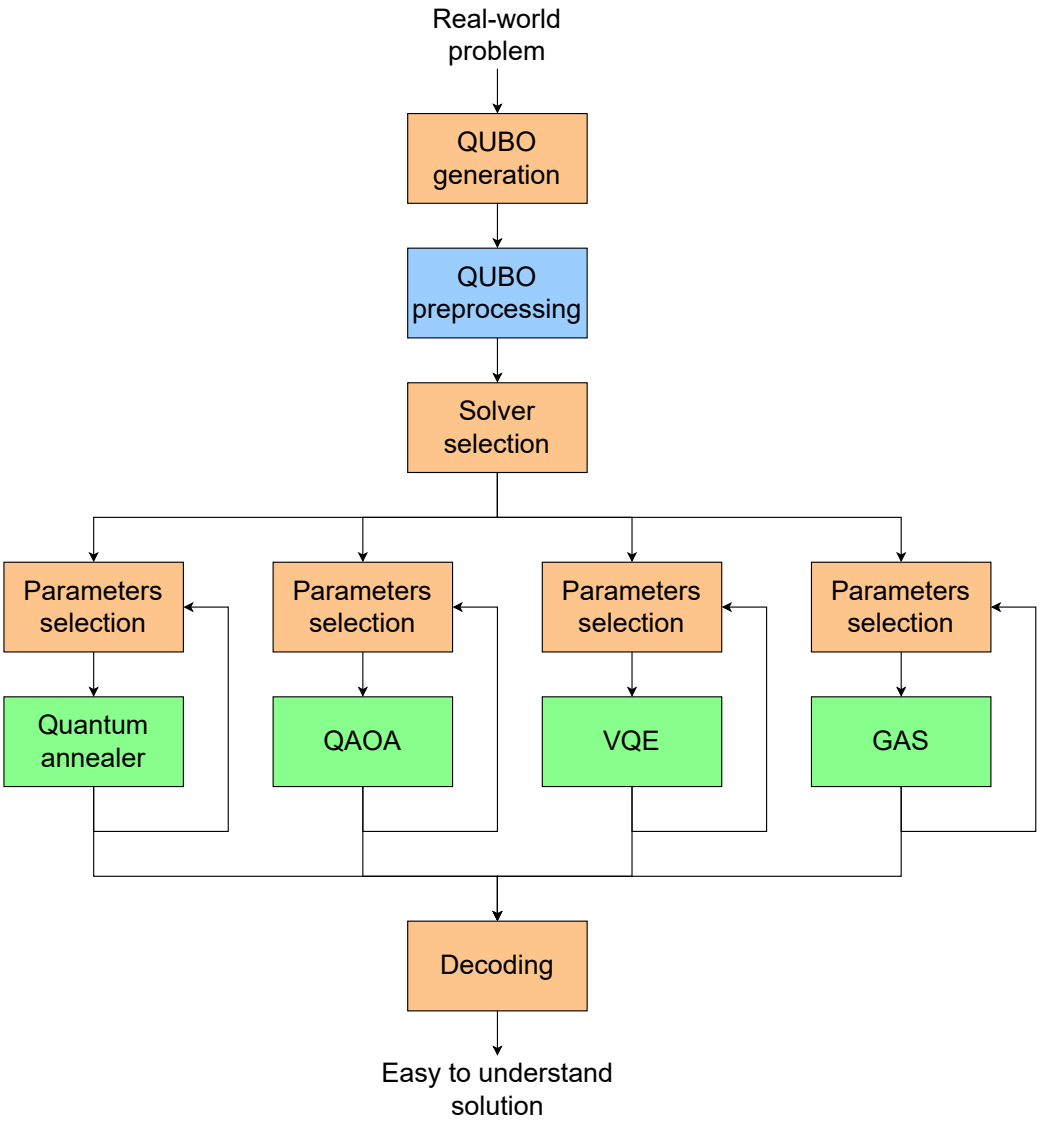


Figure 9.1: Idea of a quantum optimization toolchain structure. In blue is reported the QUBO toolchain described in this thesis. In green are highlighted the quantum solvers that can be chosen by the toolchain. The solvers indicated are just to be considered as an example and not as an exhaustive list of the available ones.

Chapter 10

Conclusions

This thesis allows speeding up the solution of combinatorial optimization problems and enhancing the exploitability of currently available quantum or quantum-assisted solvers. It tries to identify a possible improvement for the limited number of qubits available in current quantum hardware through preprocessing of cost functions expressed in the QUBO formalism. At first, an introduction to quantum computing and the Grover Adaptive Search is provided to motivate the need for a preprocessing toolchain. Afterward, an introduction to the QUBO formulation is given to understand the characteristics of quadratic binary cost functions and to explore the details of the classes of problems used as benchmarks to test the toolchain.

The goal of the toolchain is to reduce the number of variables of the cost function or decompose it into smaller ones. For this purpose, it is transformed into a network representation, hence in the thesis, for the comprehension of the toolchain behavior, an exhaustive treatment is provided of the fundamentals of these particular kinds of graphs and all the instruments and algorithms to calculate the maximum flow, obtain a residual network, find strongly connected components and disconnected components. By using all these graph tools, it is shown how persistencies can be found, thus reducing the number of variables in a cost function, and decompositions can be applied. This thesis also displays that, by extracting complete strongly connected components from the residual network associated with the cost function, it is possible to use just the Grover search algorithm to find the solution to these subproblems instead of using the more time-consuming GAS. Moreover, it is reported how to find a lower bound to the cost function allowing the estimation of the number of qubits needed to encode the values assumed by the function in a quantum state.

Finally, minimum vertex cover, maximum clique and maximum cut benchmarks have been employed to test and validate the methods explored. The results show that improvements have been achieved, i.e. the toolchain makes it possible to solve problems with larger dimensions than what only GAS can solve, especially for

QUBO functions with low densities. The lower bound estimations allow an accurate definition of the span of values assumed by the function, hence the qubits needed to encode them. However, this is still not enough to make quantum computers suitable to process all the real-world optimization problem instances. Next steps to make the employed methods more effective are also mentioned, such as the inclusion of quadratic persistencies and the use of algorithms for variable selection in the Shannon decomposition.

Furthermore, future perspectives of this thesis have been identified in the construction of a more complete toolchain able to generate the QUBO formulation of combinatorial optimization problems given as input in a user-friendly fashion, apply preprocessing to the obtained cost function and choose the best quantum solver to solve every particular problem, thus not limiting itself only to GAS.

The toolchain presented can be employed within a set of methods useful to automate the solution of optimization problems with quantum computers. It successfully reduces the number of variables of QUBO problems with low densities and it is disclosed how theoretically permits to use directly the Grover search to CSCC-free cost functions. For higher densities, the toolchain limits are highlighted, thus setting the goal of breaking and reducing complete strongly connected components as future strive to further improve the obtained results.

Current quantum hardware can not scale sufficiently to address problems, and non-idealities further limit the possibility of exploiting them in a real-world scenario. The proposed toolchain, improved with the future automation steps, could have an impact on making feasible for everyone the resolution of optimization problems whose dimensions were inconceivable until the time being.

Bibliography

[1] Samuel Palmer, Serkan Sahin, Rodrigo Hernandez, Samuel Mugel, and Roman Orus. Quantum portfolio optimization with investment bands and target volatility, 2021. https://doi.org/10.48550/ARXIV.2106.06735.

[2] Seo Woo Hong, Pierre Miasnikof, Roy Kwon, and Yuri Lawryshyn. Market graph clustering via qubo and digital annealing. *Journal of Risk and Financial Management*, 14(1), 2021. https://doi.org/10.3390/jrfm14010034.

[3] K. Giaro, M. Kubale, and P. Obszarski. A graph coloring approach to scheduling of multiprocessor tasks on dedicated machines with availability constraints. *Discrete Applied Mathematics*, 157(17):3625–3630, 2009. https://doi.org/10.1016/j.dam.2009.02.024.

[4] M. Sriram and S. M. Kang. Detailed layer assignment for mcm routing. In *Proceedings of the 1992 IEEE/ACM International Conference on Computer-Aided Design*, ICCAD '92, page 386–389, Washington, DC, USA, 1992. IEEE Computer Society Press. https://doi.org/10.1109/ICCAD.1992.279341.

[5] Yasuharu Okamoto. Maximizing gerrymandering through ising model optimization. *Scientific Reports*, 11, 2021. https://doi.org/10.1038/s41598-021-03050-z.

[6] Darrall Henderson, Sheldon H. Jacobson, and Alan W. Johnson. *The Theory and Practice of Simulated Annealing*, pages 287–319. Springer US, 2003. https://doi.org/10.1007/0-306-48056-5_10.

[7] Frank Leymann and Johanna Barzen. The bitter truth about gate-based quantum algorithms in the nisq era. *Quantum Science and Technology*, 5(4):044007, sep 2020. https://doi.org/10.1088/2058-9565/abae7d.

[8] Noson S. Yanofsky and Mirco A. Mannucci. *Quantum Computing for Computer Scientists*. Cambridge University Press, 1 edition, 2008.

[9] David P. DiVincenzo. The physical implementation of quantum computation. *Fortschritte der Physik*, 48(9-11):771–783, 2000. https://doi.org/10.1002%

2F1521-3978%28200009%2948%3A9%2F11%3C771%3A%3Aaid-prop771%3E3.0.co%3B2-e.

[10] Colin P. Williams. *Explorations in Quantum Computing*. Springer Publishing Company, Incorporated, 2nd edition, 2008.

[11] Amira Abbas, Stina Andersson, Abraham Asfaw, Antonio Corcoles, Luciano Bello, Yael Ben-Haim, Mehdi Bozzo-Rey, Sergey Bravyi, Nicholas Bronn, Lauren Capelluto, Almudena Carrera Vazquez, Jack Ceroni, Richard Chen, Albert Frisch, Jay Gambetta, Shelly Garion, Leron Gil, Salvador De La Puente Gonzalez, Francis Harkins, Takashi Imamichi, Pavan Jayasinha, Hwajung Kang, Amir h. Karamlou, Robert Loredo, David McKay, Alberto Maldonado, Antonio Macaluso, Antonio Mezzacapo, Zlatko Minev, Ramis Movassagh, Giacomo Nannicini, Paul Nation, Anna Phan, Marco Pistoia, Arthur Rattew, Joachim Schaefer, Javad Shabani, John Smolin, John Stenger, Kristan Temme, Madeleine Tod, Ellinor Wanzambi, Stephen Wood, and James Wootton. Learn quantum computation using qiskit, 2020. [online] https://qiskit.org/textbook/, accessed 24-January-2023.

[12] Lov K. Grover. A fast quantum mechanical algorithm for database search, 1996. https://doi.org/10.48550/arxiv.quant-ph/9605043.

[13] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, 2010. https://doi.org/10.1017/CBO9780511976667.

[14] Fred Glover, Gary Kochenberger, and Yu Du. A tutorial on formulating and using qubo models, 2018. https://doi.org/10.48550/arxiv.1811.11538.

[15] Kotaro Tanahashi, Shinichi Takayanagi, Tomomitsu Motohashi, and Shu Tanaka. Application of ising machines and a software development for ising machines. *Journal of the Physical Society of Japan*, 88(6):061010, 2019. https://doi.org/10.7566/JPSJ.88.061010.

[16] Martin Anthony, Endre Boros, Yves Crama, and Aritanan Gruber. Quadratic reformulations of nonlinear binary optimization problems. *Mathematical Programming*, 2017. https://doi.org/10.1007/s10107-016-1032-4.

[17] Joseph T. Iosue. Qubovert documentation. https://qubovert.readthedocs.io/en/latest/.

[18] Austin Gilliam, Stefan Woerner, and Constantin Gonciulea. Grover adaptive search for constrained polynomial binary optimization. *Quantum*, 2021. https://doi.org/10.22331/q-2021-04-08-428.

[19] Qiskit Development Team. Qiskit documentation. https://qiskit.org/documentation/.

[20] Luigi Giuffrida, Deborah Volpe, Giovanni Amedeo Cirillo, Maurizio Zamboni, and Giovanna Turvani. Engineering grover adaptive search: Exploring the degrees of freedom for efficient qubo solving. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 12(3):614–623, 2022. https://doi.org/10.1109/JETCAS.2022.3202566.

[21] Cython project. Cython documentation. [online] https://cython.readthedocs.io/en/latest/src/quickstart/overview.html, accessed 31-January-2023.

[22] Greg Ewing. Pyrex documentation. [online] https://www.csse.canterbury.ac.nz/greg.ewing/python/Pyrex/version/Doc/About.html, accessed 31-January-2023.

[23] Endre Boros, Peter Hammer, and Gabriel Tavares. Preprocessing of unconstrained quadratic binary optimization. *RUTCOR Research Report*, 2006.

[24] J.A. Bondy and U.S.R. Murty. *Graph theory with applications*. American Elsevier Publishing Company, 1976.

[25] Bernhard Korte and Jens Vygen. *Combinatorial Optimization: Theory and Algorithms*. Springer Berlin Heidelberg, 2012. https://doi.org/10.1007/978-3-662-56039-6.

[26] Andrew V. Goldberg and Robert E. Tarjan. A new approach to the maximum-flow problem. *J. ACM*, 35(4), 1988. https://doi.org/10.1145/48014.61051.

[27] B. V. Cherkassky and A. V. Goldberg. On implementing the push—relabel method for the maximum flow problem. *Algorithmica*, 1997. https://doi.org/10.1007/PL00009180.

[28] J. Cheriyan and S. N. Maheshwari. Analysis of preflow push algorithms for maximum network flow. *SIAM Journal on Computing*, 1989. https://doi.org/10.1137/0218072.

[29] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction to Algorithms, third edition*. Computer science. MIT Press, 2009. https://mitpress.mit.edu/9780262046305/introduction-to-algorithms/.

[30] Robert Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1972. https://doi.org/10.1137/0201010.

[31] Fred Glover, Bahram Alidaee, César Rego, and Gary Kochenberger. One-pass heuristics for large-scale unconstrained binary quadratic problems. *European Journal of Operational Research*, 137(2):272–287, 2002. https://doi.org/10.1016/S0377-2217(01)00209-0.

[32] Saïd Hanafi, Ahmed-Riadh Rebai, and Michel Vasquez. Several versions of the devour digest tidy-up heuristic for unconstrained binary quadratic problems. *Journal of Heuristics*, 2013. https://doi.org/10.1007/s10732-011-9169-z.

[33] Bengt Aspvall, Michael F. Plass, and Robert Endre Tarjan. A linear-time algorithm for testing the truth of certain quantified boolean formulas. *Information Processing Letters*, 1979. https://doi.org/10.1016/0020-0190(79)90002-4.

[34] Alain Billionnet and Brigitte Jaumard. A decomposition method for minimizing quadratic pseudo-boolean functions. *Operations Research Letters*, 1989. https://doi.org/10.1016/0167-6377(89)90043-6.

[35] Elijah Pelofske, Georg Hahn, and Hristo Djidjev. Decomposition algorithms for solving np-hard problems on a quantum annealer. *Journal of Signal Processing Systems*, 2021. https://doi.org/10.1007/s11265-020-01550-1.

[36] Guillaume Chapuis, Hristo N. Djidjev, Georg Hahn, and Guillaume Rizk. Finding maximum cliques on the d-wave quantum annealer. 2018. https://doi.org/10.48550/ARXIV.1801.08649.

[37] Giuseppe F. Italiano, Luigi Laura, and Federico Santaroni. Finding strong bridges and strong articulation points in linear time. *Theoretical Computer Science*, 447:74–84, 2012. https://doi.org/10.1016/j.tcs.2011.11.011.

[38] 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 25-October-2021.

[39] Alberto Moraglio, Serban Georgescu, and Przemysław Sadowski. Autoqubo: Data-driven automatic qubo generation. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, GECCO '22, page 2232–2239, New York, NY, USA, 2022. Association for Computing Machinery. https://doi.org/10.1145/3520304.3533965.

[40] Marcos Diez García, Mayowa Ayodele, and Alberto Moraglio. Exact and sequential penalty weights in quadratic unconstrained binary optimisation with a digital annealer. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, GECCO '22, page 184–187, New York, NY, USA,

2022. Association for Computing Machinery. https://doi.org/10.1145/3520304.3528925.

[41] Amit Verma and Mark Lewis. Variable reduction for quadratic unconstrained binary optimization, 2021.

[42] Fred Glover, Mark Lewis, and Gary Kochenberger. Logical and inequality implications for reducing the size and complexity of quadratic unconstrained binary optimization problems. *European Journal of Operational Research*, 05 2017. https://doi.org/10.48550/arXiv.1705.09545.

[43] Benedikt Poggel, Nils Quetschlich, Lukas Burgholzer, Robert Wille, and Jeanette Miriam Lorenz. Recommending solution paths for solving optimization problems with quantum computing, 2022. https://doi.org/10.48550/arXiv.2212.11127.

[44] Nils Quetschlich, Lukas Burgholzer, and Robert Wille. Towards an automated framework for realizing quantum computing solutions, 2023. https://doi.org/10.48550/arXiv.2210.14928.