Quantum Machine Intelligence (2024) 6:48
https://doi.org/10.1007/s42484-024-00179-8

RESEARCH ARTICLE

Analyzing the effectiveness of quantum annealing with meta-learning

Riccardo Pellini[1] · **Maurizio Ferrari Dacrema**[1]

Received: 14 April 2024 / Accepted: 10 July 2024

Abstract
The field of Quantum Computing has gathered significant popularity in recent years and a large number of papers have studied its effectiveness in tackling many tasks. We focus in particular on Quantum Annealing (QA), a meta-heuristic solver for Quadratic Unconstrained Binary Optimization (QUBO) problems. It is known that the effectiveness of QA is dependent on the task itself, as is the case for classical solvers, but there is not yet a clear understanding of which are the characteristics of a problem that make it difficult to solve with QA. In this work, we propose a new methodology to study the effectiveness of QA based on meta-learning models. To do so, we first build a dataset composed of more than five thousand instances of ten different optimization problems. We define a set of more than a hundred features to describe their characteristics and solve them with both QA and three classical solvers. We publish this dataset online for future research. Then, we train multiple meta-models to predict whether QA would solve that instance effectively and use them to probe which features with the strongest impact on the effectiveness of QA. Our results indicate that it is possible to accurately predict the effectiveness of QA, validating our methodology. Furthermore, we observe that the distribution of the problem coefficients representing the bias and coupling terms is very informative in identifying the probability of finding good solutions, while the density of these coefficients alone is not enough. The methodology we propose allows to open new research directions to further our understanding of the effectiveness of QA, by probing specific dimensions or by developing new QUBO formulations that are better suited for the particular nature of QA. Furthermore, the proposed methodology is flexible and can be extended or used to study other quantum or classical solvers.

Keywords Quantum Computing · Quantum Annealing · Optimization · Meta-learning

1 Introduction

In recent years, the field of Quantum Computing has gathered significant popularity, thanks to remarkable advancements that led to the development of several quantum computers of different architectures and technologies that can be used to tackle numerous problems. Although quantum computers are still limited both by their relatively small size and by the noise that limits the precision of the computation, the field is rapidly moving forward. Among the existing Quantum Computing paradigms, Quantum Annealing (QA) is a meta-heuristic that can be used to solve Quadratic Unconstrained Binary Optimization (QUBO) problems, a family of NP-hard optimization problems. The key idea of QA is to represent a QUBO problem as an energy minimization problem of a real and configurable quantum device. To do so, the problem variables are mapped onto physical quantum bits or qubits. The quantum device is steered toward a state of minimal energy, called *ground state*, with a controlled evolution. The ground state corresponds to the optimal solution of the original QUBO problem. The devices that implement the QA process are called *Quantum Annealers*.

The ability of QA to tackle NP-hard optimization problems and its flexibility to heterogeneous domains is what makes it an interesting technology for industries and researchers. Many applications of QA have been proposed in the fields of machine learning (Neukart et al. 2018; Mott et al. 2017; Mandrà et al. 2016; Ferrari Dacrema et al. 2022; Neven et al. 2009; Willsch et al. 2020; Kumar et al. 2018; Neukart

✉ Riccardo Pellini
riccardo.pellini@polimi.it

Maurizio Ferrari Dacrema
maurizio.ferrari@polimi.it

[1] Politecnico di Milano, Milan, Italy

Published online: 25 July 2024

et al. 2018; O'Malley et al. 2017; Ottaviani and Amendola 2018; Nembrini et al. 2021), chemistry (Micheletti et al. 2021; Hernandez and Aramon 2017; Streif et al. 2019; Xia et al. 2018) and logistics (Ikeda et al. 2019; Rieffel et al. 2015; Ohzeki 2020; Stollenwerk et al. 2017), but the results are not always competitive against classical heuristics solvers.

An important issue is that the quality of the solutions found by QA is limited by multiple factors. First of all, Quantum Annealers are physical devices that have a limited number of qubits and connections between them. This limits the size of the problems that they can tackle and requires to process of the QUBO problem adapting it to the physical structure of the Quantum Annealer. A second important aspect is that the quality of the solutions found by QA depends on the behavior of the underlying physical quantum system, which is very difficult to study. It is known that some problems appear to be more difficult to solve with QA (Yarkoni et al. 2022; Jiang and Chu 2023; Huang et al. 2023), but understanding why is not a trivial task and still an open research question.

Most of the previous studies on QA compare its performance, in terms of required time for computation with respect to other heuristic solvers, rather than on the quality of the solutions it finds, i.e., its effectiveness. There are two ways in which one can study the effectiveness of QA, one is by analytically describing the underlying quantum behavior and the other is to perform empirical experiments. A theoretical analysis has been performed for very small QUBO instances (Stella et al. 2005), which, however, are too simple to assess the effectiveness of QA when compared to other classical solvers. Furthermore, analytically analyzing such a quantum system becomes rapidly very expensive and is generally impossible for problems of interesting size. On the other hand, the existing empirical studies on the effectiveness of QA have explored much larger problems but focus mainly on specific tasks such as feature selection (Ferrari Dacrema et al. 2022), clustering (Kumar et al. 2018; Neukart et al. 2018) and classification (Mott et al. 2017; Willsch et al. 2020; Neven et al. 2009), and therefore, lack generality.

To the best of our knowledge, there is no published research that has investigated extensively how the characteristics of the problem impact the effectiveness of QA. For this reason, in this study, we propose a novel empirical methodology for the analysis of the effectiveness of QA, based on the study of the characteristics of QUBO problems with a meta-learning approach. The general idea consists of generating many QUBO instances, defining a set of features that can describe them, and train meta-models to predict whether QA would solve that problem or not. Our key contributions are as follows:

- The design of an experimental methodology that can be applied to study the effectiveness of QA. This methodology can be used also for other quantum algorithms, such as QAOA (Farhi et al. 2014) or VQE (Fedorov et al. 2022);
- The selection of ten classes of optimization problems, each one with specific characteristics, from which we generate approximately five thousand QUBO instances;
- The design and the generation of a meta-learning dataset, which contains for each of the five thousand instances a selection of a hundred features based on probability theory, statistics, and graph theory. We show that using them it is possible to effectively predict whether QA would solve a problem instance effectively or not. We share the meta-learning dataset online for further research;
- The analysis of the features of a QUBO problem with the strongest impact on the effectiveness of QA;

2 Background

2.1 QUBO and Ising models

In order to use Quantum Annealing (QA) to tackle optimization problems, these should be represented with one of two equivalent formulations called QUBO and *Ising*, suitable for NP-Complete, and some NP-Hard optimization problems (Glover et al. 2022; Lucas 2014). While the two are equivalent, the QUBO formulation is closer to traditional Operations Research, the Ising formulation is instead closer to Physics.

The objective function in the QUBO model is given by Eq. 1, where $x \in \{0, 1\}^n$ is a column vector representing the *assignment* of the binary variables $x_1, x_2, ..., x_n$, n is the number of problem variables, y the cost, and $Q \in \mathbb{R}^{n \times n}$ is a real square matrix, either symmetric or upper triangular.

$$\min_x y = x^T Q x \tag{1}$$

We will refer to combinatorial optimization problems written in the QUBO formulation as *QUBO problems*. Note that the QUBO formulation does not allow for hard constraints. An optimization problem with constraints can be transformed into a QUBO problem by introducing a quadratic penalty term multiplied by a penalty coefficient p. The idea is that the hard constraints are transformed in *soft* constraints, such that if they are violated a positive penalty p is added to the cost function making the cost of that variable assignment worse. Note that by using soft constraints we do not have the guarantee that the optimal solution will satisfy the constraints, which may happen frequently if the penalty coefficient p has a value that is too low. In general, a quadratic binary optimization problem with equality constraints formulated as $Ax - d = 0$, where $d \in \mathbb{R}^m$ and $A \in \mathbb{R}^{m \times n}$, can

be transformed into the following QUBO problem:

$$\min_x y = x^T Qx + p \cdot x^T Cx \quad (2)$$
$$C = (Ax - d)^T (Ax - d)$$

If the quadratic binary optimization problem also has inequality constraints, those need to be transformed first into equality constraints using *binary slack variables*. For example, if we have the following constraint:

$$x_1 + 2x_2 + 4x_3 \leq 3$$

we can transform it into an equality constraint by introducing the binary slack variables x_4 and x_5:

$$x_1 + 2x_2 + 4x_3 + x_4 + 2x_5 = 3$$

There exist no general rule to choose the best number of slack variables, so multiple strategies can be followed.

A second useful formulation is the *Ising model*, which was developed to describe an energy minimization problem for a system of particles (Glover et al. 2022; Lucas 2014). The objective function of the Ising model is given by Eq. 3, where $s \in \{-1, 1\}^n$ is the column vector representing the assignment of the n problem variables $s_1, s_2, ..., s_n$ also called *spin* variables, $J \in \mathbb{R}^{n \times n}$ is the *coupling* matrix that describes the quadratic terms of the objective function and has zero diagonal, $b \in \mathbb{R}^n$ is the bias vector, which contains the linear terms of the objective function. The constant term $c \in \mathbb{R}$ is called offset.

$$\min_s y = s^T Js + b^T s + c \quad (3)$$

A QUBO problem can be transformed into an Ising problem through a linear mapping of the variables. In particular, a binary variable x_i is transformed into a spin variable s_i according to the following conversion[1]:

$$x_i = \frac{1 - s_i}{2}$$

2.2 Quantum Annealing and Quantum Annealers

Quantum Annealing (QA) is a meta-heuristic solver for QUBO problems. It is based on the Adiabatic Quantum Computation (AQC) paradigm, with some relaxations (Yarkoni et al. 2022; Morita and Nishimori 2008; Farhi et al. 2000; Albash and Lidar 2018; Hauke et al. 2020). The idea is to represent the optimization problem as an energy minimization one, and then use a configurable device that exhibits the needed quantum behavior to minimize it. Such a device, the Quantum Annealer, is composed of multiple *qubits* connected between each other. QA works based on a time evolution of the quantum system. The initial state of the system is a default one, easy to prepare, so that the qubits are in a state of minimal energy, i.e., the *ground state*. Then, the physical system evolves slowly over a short amount of time by introducing a dependency on the Ising coefficients of the problem one wishes to solve. This means, for example, slowly changing the magnetic fields the qubits are subject to. At the end of the evolution, the physical system will depend only on the problem and, if the evolution was careful enough, it will still be in the ground state. Since the state of minimal energy is also the solution to the optimization problem, measuring the state of the qubits will yield the values that the problem variables should have.

The evolution of the system in QA occurs in a noisy environment and is subject to quantum fluctuations, i.e., *quantum tunneling*, which helps it explore the solution space. The noise of the system and the duration of the evolution influence the results of QA, if the evolution is too fast the system will likely escape its ground state and find a worse solution, while if the evolution is too slow noise may build up and push the system again out of the ground state. Due to its stochastic nature, QA acts as a device sampling low-cost solutions in a similar way as other classical solvers do, such as Simulated Annealing. For this reason, QA is repeated multiple times in order to obtain *samples* of the final state of the quantum system.

The physical devices that implement QA are called *Quantum Annealers*. Currently, D-Wave Systems Inc. is the company that provides the Quantum Annealers with the largest number of qubits.[2] For example, the D-Wave Advantage has more than 5000 qubits with a topology called Pegasus, where each qubit is connected to other 15 ones.

Solving a QUBO problem with a Quantum Annealer requires the following steps:

1. **Formulate the problem as a QUBO or an Ising problem**: the coefficients that are needed to configure the Quantum Annealers are those of the Ising formulation, as such the problem needs to be in this form. If the problem has a simpler formulation as a QUBO, the transformation is straightforward. Note that some problems can be formulated as QUBO or Ising easily, while others require more expensive processing.
2. **Embed the problem on the topology of the device:** since the Quantum Annealer is a physical object, we

[1] This mapping is equivalent to the more commonly used $x_i = \frac{1+s_i}{2}$ with the difference that in our case a binary 0 is mapped onto spin 1 and binary 1 is mapped onto spin -1.

[2] The documentation of the D-Wave Systems' services: https://docs.ocean.dwavesys.com/en/stable/index.html

must fit the problem we want to solve on it, accounting for the limited number of qubits and of the connections between them. This procedure is called *minor embedding* (Carmesin 2022) and maps each problem variable to one or more qubits. If multiple qubits are needed to represent a single problem variable, that is called a qubit *chain*. If the problem has a large number of quadratic terms, a substantial number of qubits may be needed to create all the physical connections. Figure 1 shows an example of how a simple problem can be mapped on a Quantum Annealer. Minor embedding is an NP-Hard problem but polynomial-time heuristic algorithms are available (Choi 2008; Cai et al. 2014; Boothby et al. 2020)[3].

3. **Evolution of the system and sampling of the solutions:** once the minor embedding is done, the problem is transferred to the Quantum Annealer. First, the device is programmed with the problem coefficients, then we can perform a sequence of multiple evolutions to obtain the desired number of samples n_s. Each sample requires three steps: (*i*) the evolution is run for the desired duration, called *annealing time* t_a, (*ii*) the final state of the system is measured, which requires a read-out time t_r dependent on the number of qubits used, and (*iii*) the device pauses shortly for cooling.

More formally, the energy of a system can be modeled with an *Hamiltonian*, $H \in \mathbb{R}^{2^n \times 2^n}$, and the evolution that occurs in QA is described by the time-dependent Hamiltonian $H(t)$ that models the transition from the initial default Hamiltonian H_i.[4] and the Hamiltonian describing the problem H_p:

$$H(t) = A(t)H_i + B(t)H_p \tag{4}$$

The coefficient $A(t)$ decreases as the evolution progresses, while $B(t)$ increases introducing the dependency on the characteristics of the problem, but their exact values depend on the hardware. At the beginning of the evolution $B(t)$ is zero, while at the end $A(t)$ is zero. Note that this is just a description of the underlying physical system and there is no need to compute this representation to use QA.

In the ideal Adiabatic Quantum Computing setting, it is possible to compute the exact annealing time needed to ensure the system remains in the ground state and finds the global optimum, this result dates back from a century ago (Born and Fock 1928). This optimal annealing time is inversely proportional to the smallest difference between the two smallest eigenvalues $\lambda_1(t), \lambda_2(t)$ of $H(t)$. Such difference is called *minimum gap*. Although this result may be useful to understand its behavior, it is not applicable to QA because it is subject to noise. Furthermore, computing the eigenvalues of $H(t)$ is prohibitive for all but the smallest problems.

To exemplify how this representation works, assume to have an Ising problem of n variables with coupling J and bias b, H_p is a $2^n \times 2^n$ matrix computed as follows:

$$H_p = \sum_{i=1}^{n} \sum_{j=i}^{n} J_{ij} \sigma_z^{(i)} \sigma_z^{(j)} + \sum_{i}^{n} h_i \sigma_z^{(i)} \tag{5}$$

The matrix $\sigma_z^{(i)}$ is the Z-Pauli operator σ_z acting on qubit i:

$$\sigma_z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \tag{6}$$

$$\sigma_z^{(i)} = \bigotimes_{k=1}^{i-1} I \otimes \sigma_z \bigotimes_{k=1}^{n-i} I \tag{7}$$

with $\otimes$ being the tensor product and I the identity matrix. A useful property of H_p is that it is a diagonal matrix that contains all the cost values for all possible variable assignments of the problem. Since it is diagonal, these values are also its eigenvalues and the corresponding eigenvectors encode the variable assignment that has that cost. The minimum eigenvalue of H_p corresponds to the minimal cost and the corresponding eigenvector to the optimal variable assignment.

As an example, consider the following QUBO problem, which is minimized when $x_1 = x_2$:

$$\min_{x_1, x_2} y = x_1 + x_2 - 2x_1x_2$$

The equivalent Ising formulation is as follows:

$$\min_{s_1, s_2} y = \frac{1}{2} - \frac{1}{2} s_1 s_2$$

For this small instance, we can compute H_p easily. The matrices $\sigma_z^{(1)}$ and $\sigma_z^{(2)}$ are:

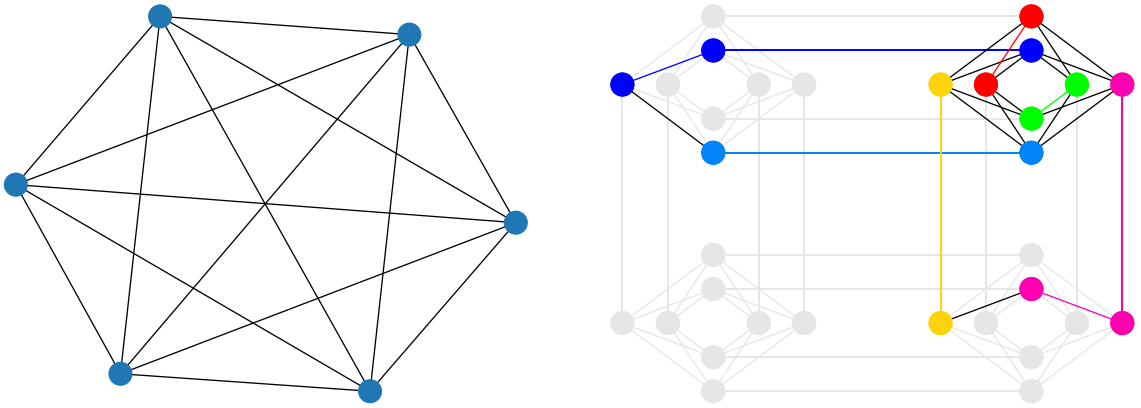
$$\sigma_z^{(1)} = \sigma_z \otimes I = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix} \quad \sigma_z^{(2)} = I \otimes \sigma_z = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix}$$

H_p is then equal to:

$$H_p = \frac{1}{2} \begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix}$$

[3] We use the library `minorminer` offered by D-Wave: https://docs.ocean.dwavesys.com/en/stable/docs_minorminer/source/sdk_index.html

[4] The initial Hamiltonian H_i is also called *driver Hamiltonian*.



(a) Problem as a graph G of six variables

(b) Embedding of G on Chimera topology

Fig. 1 Embedding of a simple problem with six variables on a portion of a D-Wave Quantum Annealer using the Chimera topology. Each node represents a qubit and each edge is a physical connection between them. Nodes of the same color indicate the chain of qubits used to represent a single problem variable. Note how, while the original problem had six variables, the embedded one requires 14 qubits

The smallest eigenvalue of H_p is $\lambda_{\min} = -\frac{1}{2}$ and has a multiplicity of two corresponding to the first and last eigenvalues. Indeed, both $x_0 = 0, x_1 = 0$ and $x_0 = 1, x_1 = 1$ are optimal solutions to the problem. If we sum $\lambda_{\min}$ with the offset of the Ising problem, $\frac{1}{2}$, we obtain 0, that is the same value of the QUBO cost function when $x_1 = x_2$.

2.3 Studies on the effectiveness of QA

Most of the previous studies on QA focus on its performance, by measuring the time required to solve a problem and comparing it to that of classical solvers. To the best of our knowledge, there is no consensus on whether QA provides a general and consistent speedup compared to other traditional solvers for QUBO problems (Hauke et al. 2020; Yarkoni et al. 2022; Katzgraber et al. 2014), while a recent paper claims substantial speedup for a quantum simulation task (King et al. 2024). Although some papers may claim a speedup, this is often based on measurements that only account for part of the process. Indeed, one should consider the time required by all phases: (*i*) formulating the optimization problem as QUBO or Ising, (*ii*) embedding the problem on the QA, (*iii*) sampling the solutions on the device and (*iv*) postprocessing the results if needed (for example by checking if the constraints are satisfied). Frequently, the efficiency of QA is measured by only accounting for the usage of the quantum device itself (programming time and the repeated annealing and read-out) while ignoring the time needed for minor embedding and for creating the QUBO formulation. This gives an incomplete picture of the technology that does not account for two significant bottlenecks. For example, it may be that in a certain situation, QA is faster than other traditional methods in solving a specific QUBO problem, but that may not be the case anymore if one includes the minor embedding phase. Furthermore, if it is very computationally expensive to formulate the problem as QUBO, it may be more efficient to use other traditional methods that do not need a QUBO formulation at all.

When comparing the quality of the solutions found by QA and classical solvers, i.e., their effectiveness, the published literature usually focuses on problems related to specific fields or even to very specific instances of those problems. Due to this, there is still a limited understanding of how would QA compare in a more general setting. For example, the effectiveness of QA has been analyzed for feature selection (Ferrari Dacrema et al. 2022), classification (Mott et al. 2017; Willsch et al. 2020; Neven et al. 2009) and clustering (Neukart et al. 2018; Kumar et al. 2018), which are typical machine learning tasks. In the field of chemistry, QA has been applied and analyzed to find the equilibria of polymer mixtures (Micheletti et al. 2021), to find similarities between molecules (Hernandez and Aramon 2017) and to find their ground state (Streif et al. 2019). The effectiveness of QA in solving problems related to logistics has been analyzed too, for example in solving the Nurse Scheduling Problem (Ikeda et al. 2019) and in optimizing the assignments of the gates at the airport (Stollenwerk et al. 2017).

Previous research also studied the effectiveness of QA from a theoretical perspective by representing analytically the evolution of the time-dependent Hamiltonian $H(t)$ (see Eq. 4) and computing the probability of escaping the ground state (Stella et al. 2005). This approach is, however, limited by the fact that the size of the Hamiltonian grows exponentially on the number of QUBO problem variables n, and the analytical analysis of the Hamiltonian becomes rapidly impractical for all but the smallest problems. An alternative

way is to adopt an empirical approach, by using the outcome of multiple experiments to probe the underlying physical system (Irsigler and Grass 2022). The idea is to allow the evolution to progress up to a certain intermediate stage and then drastically accelerate it (i.e., a quench, according to D-Wave terminology), observing how the effectiveness changes based on when the evolution accelerated. While this approach allows to tackle of large problem instances, applying the acceleration at different stages of the evolution requires to repeat the experiment a large number of times and therefore this approach too is very resource-intensive.

To overcome the limitations of the methods adopted in the literature, this paper proposes a new empirical approach to study how the quality of the solutions found by QA is impacted by the characteristics of the problem. To achieve this, we first collect a dataset of problem instances belonging to 10 selected problem classes and solve them using both QA and three classical solvers. Then, for each problem instance, we compute a set of features describing various characteristics, from the distribution of the bias coefficients of its QUBO formulation to the topology of the graph that describes the instance once it has been embedded in the Quantum Annealer. Using this dataset, we train a machine learning classifier to identify whether QA was able to find a good solution for that instance and, finally, use it to assess which are the most important problem features.

3 Meta-learning dataset generation

In this section, we present the methodology used to generate our meta-learning dataset, on which we train the meta-models to predict the effectiveness of QA. We publish this dataset online for future research. First, we describe how we select the ten classes of problems we want to solve with QA and the strategies we use to generate the five thousand instances. Then, we describe how we evaluate the effectiveness of QA, in terms of closeness to the optimal solution of the problem and by comparing QA with other classical methods (Simulated Annealing, Tabu Search, and Steepest Descent). Third, we describe the representations of the QUBO problem we used to compute the approximately one hundred features used to train the meta-models. Finally, we describe how we solve the instances with QA and with the classical solvers, with a particular focus on the choice of the optimal hyperparameters of the solvers.

3.1 Selection of problems and instances

We identify a selection of ten different optimization problems that exhibit different characteristics: some have constraints, others do not; some have linear terms, others do not; some have a large number of quadratic terms while others do not, etc. The details on their formulations are reported in Appendix 1 and the details on the generation of the instances are in Appendix 2.

The first group contains five classes of optimization problems defined over a graph: Max-Cut, Minimum Vertex Cover, Maximum Independent Set, Max-Clique, and Community Detection. They were selected for the following reasons. Both the Max-Cut and Community Detection problems have a straightforward QUBO formulation that does not require penalties to represent constraints. The Max-Cut, Maximum Independent Set, and Minimum Vertex Cover problems share the same quadratic terms in their QUBO matrix, but not the diagonal (i.e., the linear terms or bias). The Max-Clique problem is formulated as a Maximum Independent Set problem but is defined on the complement graph. The Community Detection problem has a very dense QUBO matrix as there are quadratic terms between all variables and is a relevant problem in Machine Learning (Nembrini et al. 2022). Since these problems are formulated on a graph, we apply them to four different graph topologies: Erdös-Renyi, Cyclic, Star, and 2d-grid. Note that in order to have a diversified set of instances we introduce small random perturbations to each topology, consisting of few edge insertions and deletions. The number of insertions and deletions depends on the number of nodes on the graph. More details are reported in Appendix 2.1.

The second group of five optimization problems contains Number Partitioning, Quadratic Knapsack, Set Packing, Feature Selection, and 4×4-Sudoku. These are a more heterogeneous set than the previous graph-based problems and so require ad-hoc strategies to generate their instances which we detail in Appendix 2.2. Similarly to the Max-Cut and Community Detection problems, the Number Partitioning problem has a straightforward QUBO formulation with no penalty terms to represent constraints. Similarly to the Community Detection problem, the Feature Selection problem has a dense QUBO matrix with quadratic terms between all variables. Finally, the Quadratic Knapsack, Set Packing, and 4×4-Sudoku problems are all Constraint Satisfaction Problems, each with different types of constraints. In particular, the Quadratic Knapsack problem has inequality constraints that need to be converted into equality constraints using slack variables.

We generate multiple instances of all the problem classes we selected. Concerning the size of the problem instances, measured in the number of problem variables, there are two constraints to take into account. First, the D-Wave Quantum Annealer that we use has more than 5000 qubits but, due to their limited connectivity, it is generally possible to tackle problem instances up to between 100 and 200 variables depending on the structure of the QUBO problem. This is due to the minor-embedding phase. Second, we want the instances to be representative of problems that are not trivial

and with a Hamiltonian that could not be analyzed analytically. In order to provide a more complete picture, we are also interested to assess the impact of the distribution of the solution space of the problem. This, formally, corresponds to the set of eigenvalues and eigenvectors of the Hamiltonian of the problem H_p (see Section 2.2). Unfortunately, it is impractical to compute them for instances of more than 32 problem variables, which may be too small and easy to allow a comparison of the effectiveness of different solvers. For these reasons, we decided to create two separate sets of instances:

- One set of *large instances*, with 5114 instances of between 69 and 99 variables, the upper range of what can be tackled with the QA;
- One set of *small instances*, with 246 instances of between 27 and 32 variables. With this set of instances, we can do a more complete analysis which includes also the distribution of the solution space.

The number of problem instances for each optimization problem is summarized in Table 1. Notice that the instances of the 4×4-Sudoku problem are included only in the small instances set since the largest possible instance is unique and it has at most 64 variables. All the generated instances are satisfiable and, when needed, the penalty term coefficient p used in the QUBO formulation is optimized with a Bayesian Search[5] (Victoria and Maragatham 2021; Snoek et al. 2012), in order to maximize the number of feasible solutions for Simulated Annealing.

3.2 Evaluating the effectiveness of a solver

In this section, we describe how to evaluate the effectiveness of a solver and, in particular, of QA. Both QA and the traditional solvers, we compare it to are stochastic and are executed multiple times to obtain a set of variable assignments that aim to minimize the cost function, which we call a set of *samples*. A sample is represented by an assignment of the decision variables x and by the related *cost value* y.

We solve all instances with QA, Simulated Annealing (SA), Tabu Search (TS), and Steepest Descent (SD). Our definition of how much a solver is *effective* is based on whether it finds samples that meet some quality constraints. While for the small instance set it is possible to compute the global optimum, for the large ones it is not feasible to do so, and therefore, we define the effectiveness in relative terms with respect to the other solvers.

[5] The range of possible values of p, for a particular instance, depends on the cost function of that instance. More details are given in Appendix 2

In particular, we evaluate the effectiveness of QA on the large instances set by comparing its samples with those of the traditional heuristic solvers. We define the samples associated with the best cost value for a solver S as $y^S_{\min}$. An instance I is *QA-over-all* if the best solution found by QA is at least as good as the best one found by SA, SD, and TS combined. More formally, if $y^{QA}_{\min} \leq \min\{y^{SA}_{\min}, y^{TS}_{\min}, y^{SD}_{\min}\}$. Comparing QA with a pool of multiple solvers results in a stricter evaluation of its effectiveness, but the condition that QA has to be at least as good as all the other solvers combined may be too strict. For this reason, we also compare QA with each individual solver. An instance I is *QA-over-S* if the best solution found by QA is at least as good as the one found by solver S, hence $y^{QA}_{\min} \leq y^S_{\min}$.

For the small instances, we can perform a deeper analysis of the effectiveness because we can explore the full solution space and find the global optimum. This is in practice done by computing the Hamiltonian of the problem, H_p, which is a diagonal matrix enumerating the eigenvalues λ, sometimes called *energy*, of all the variable assignments. The eigenvalue is equivalent to the cost function y but does not include possible constant offsets c from the Ising formulation, therefore $y = \lambda + c$. The variable assignment x associated with an eigenvalue λ can be computed starting from the corresponding eigenvector of H_p. The global optimum of an instance is the assignment $x_{\min}$ corresponding to the minimal eigenvalue of H_p, $\lambda_{\min}$. We will refer to the maximum eigenvalue as $\lambda_{\max}$.

We define a sample with energy λ as ϵ-*Optimal* if the following condition holds:

$$\lambda \leq \lambda_{\min} + \epsilon \cdot (\lambda_{\max} - \lambda_{\min}) \tag{8}$$

The ϵ-Optimality condition describes how close is the eigenvalue of a sample to the solution of the instance. The coefficient $\epsilon \in [0, 1]$ allows to restrict the interval under which λ is considered close enough to the optimal eigenvalue $\lambda_{\min}$. Notice that if $\epsilon = 0$ only the global optimum of the instance meets the constraint in Eq. 8.

We also define a sample x as *Hamming-Optimal* (h-Optimal) if it differs from any solution $x_{\min}$ in at most one decision variable. This corresponds to check the Hamming distance between a sample and a solution of the instance:

$$||x - x_{\min}||_{\text{Hamming}} \leq 1 \tag{9}$$

3.3 Meta-learning features

In this section, we introduce the features we define to describe a problem instance. We rely on a selection of metrics used in statistics and probability theory, such as the Gini coefficient (Damgaard and Weiner 2000), the Herfindahl-Hirschman index (Brezina et al. 2016) and the Shannon entropy (Shan-

Table 1 Each row of the table gives the number of small and large instances related to each problem class

	Problem class	Number of instances	
		Small	Large
Graph	Max-Cut	20	620
	Minimum Vertex Cover	20	619
	Maximum Independent Set	20	620
	Maximum Clique	20	620
	Community Detection	20	620
No-graph	Quadratic Knapsack	20	620
	Set Packing	20	620
	Number Partitioning	20	620
	Feature Selection	20	155
	4 × 4-Sudoku	30	–

non 1948), as well as metrics used in graph theory, such as the spectral gap, the radius a graph, its diameter and its connectivity. In total, we compute 107 features, which we describe in detail in Appendices 3 and 4.

The features we compute can be grouped in multiple domains of analysis. Overall, we identify seven domains, among which we describe the three most relevant ones:

- **Logical Ising Graph (LogIsing):** This domain uses the Ising formulation of a QUBO problem. It is represented as a graph having one node per problem variable, associated with the corresponding bias b, and an adjacency matrix that corresponds to the coupling matrix J;
- **Embedded Ising Graph (EmbIsing):** This domain uses the Ising formulation of a QUBO problem obtained after its minor embedding on the QA. The target architecture is D-Wave Advantage with the Pegasus topology. Therefore, this formulation represents the actual problem solved by the Quantum Annealer, in which multiple qubits may be used to represent one problem variable. This formulation is represented as a graph in the same way as LogIsing;
- **Solution Space (SolSpace):** This domain uses the eigenvalues, or energy values, of all possible variable assignments, which can be computed only for the small instances, and aims to describe how they are distributed.

Other domains we identify are (*i*) Normalized Multiplicity (NorMul), whose features are related to the multiplicity of the eigenvalues of H_p; (*ii*) Matrix Structure (MatStruct), which contains features related to the distribution of the values of the matrix Q of a QUBO problem; (*iii*) 25%-SolSpace and 25%-NorMul, which contain the same features of SolSpace and NorMul, but computed by considering only the 25% lowest eigenvalues of H_p, i.e., the energies of the 25% best solutions.

For the LogIsing and EmbIsing domains, we compute several features on different mathematical objects, such as the coupling matrix J, the Laplacian matrix of the corresponding graph, and the bias vector. We call such objects *components*.

We can also identify sets of features that refer to the same mathematical object but are computed on different domains. For example, both LogIsing and EmbIsing domains include features computed on the bias. We refer to them as *component sets* and allow us to perform an analysis of the importance of those mathematical objects that is orthogonal to that of the domains. We identified the following component sets: Coupling, Bias, Laplacian, Structural Adjacency (StructAdj), and Structural Laplacian (StructLap), where StructAdj and StructLap gather features related to the binarized versions of the coupling and Laplacian matrices.

3.4 Hyperparameter optimization of the solvers

Since the goal of this study is to compare the effectiveness of different solvers, it is essential to ensure that each solver is using the best hyperparameters. Indeed, it is well known in many fields that comparing methods that are not consistently optimized leads to inconsistent results that cannot be used to draw reliable conclusions (Shehzad and Jannach 2023; Ferrari Dacrema et al. 2021). The same applies in our case.

We optimize the hyperparameters of each solver (QA, SA, TS, SD) on the instance with the largest minor embedding on D-Wave Quantum Annealer for each optimization problem class. The goal is to identify the hyperparameters that will lead the solver to find the variable assignment with the lowest cost y. Once the optimal hyperparameters have been found, they are used to solve all instances of the corresponding problem class. We optimize separately the hyperparameters used for the large and small instance sets. To optimize the hyperparameters of the classical solvers, we use the standard QUBO formulation, while for QA, we use the embedded QUBO formulation: we followed this strategy because the embedded QUBO formulation is required only for QA. In this way, we have a fair comparison between different solvers since, for

each one of them, we take into account only the necessary steps to solve a QUBO instance.

Optimal hyperparameters of Quantum Annealing

QA has several hyperparameters that can be optimized,[6] some of which refer to the evolution process as a whole while others allow to fine-tune it at the level of each individual qubit. The access to the D-Wave Quantum Annealers is limited and for such a large set of instances, we have devised a methodology to optimize the hyperparameters we believe are the most important: the *annealing time* t_a and the *number of samples* n_s. In order to perform an efficient optimization within the available resources, we define a fixed computational budget T for each instance. Using the default annealing time, $20\mu s$, and drawing 100 samples requires, in the worst case, at most 37 ms. In our experiments, we allocated $T = 70\ ms$ and $T = 300\ ms$ per each problem instance.

The optimization is performed by iterating over 10 values for t_a, approximately equidistant from each other, between 5 μs and 200 μs. Given t_a, the number of samples n_s is computed as the maximum value allowed within the computational budget T, according to Eq. 10. For $T = 70\ ms$, n_s is between 145 and 537 while, for $T = 300\ ms$, n_s is between 766 and 2826.

$$n_s = \left\lfloor \frac{T - t_p}{t_a + t_r + \Delta} \right\rfloor \tag{10}$$

The term $t_p \simeq 15\ ms$ is the time needed to program the instances on the Quantum Annealer, t_r is the read-out time, needed to read the results of the annealing process, and $\Delta \simeq 20\ \mu s$ is the delay applied after each read-out operation. The read-out time t_r is unknown a-priori because it depends on the size of the embedded problem. Based on empirical observation, we use $t_r = 75\ \mu s$ for small instances and $t_r = 150\ \mu s$ for large instances. We choose t_a and the related n_s which provide the sample with the lowest energy. If for multiple pairs (t_a, n_s) QA finds samples with the lowest energy, we choose the pair with the smallest t_a.

Since the results we obtained when using both computational budgets are very similar, we report those for $T = 70\ ms$. The selected hyperparameters are reported in Table 2. Notice that the annealing time t_a for the large instances is often smaller than for the small instances. This highlights that, for large instances, a larger t_a does not improve the effectiveness of QA, at least in the range of values, we considered and for the number of samples it allows to draw. Such result suggests that QA may require an additional optimization, for example, of the annealing schedule, which is not straightforward and it goes beyond the scope of this study.

Optimal hyperparameters of the classical solvers

The hyperparameters of Simulated Annealing (SA), Tabu Search (TS), and Steepest Descent (SD) are optimized with the following procedure. For the optimization of these methods, we do not use a fixed computational budget because the technology is fundamentally different and, due to the various stages required by QA, it is not trivial to define such a comparison in a way that is *fair*. First, we fix the number of samples to $n_s = 200$ which is a value comparable to that used for QA. For half of the large instances, QA uses more samples than the classical solvers, while for the remaining half the opposite is true. We optimize the hyperparameters with a Bayesian Search of 100 iterations (Victoria and Maragatham 2021; Snoek et al. 2012). The results are available in Appendix 5. For TS, we optimize the number of restarts of the algorithm and the initialization strategy. For SD, there are no hyperparameters to optimize, except for the number of samples, which we have already set. For what concerns SA, we optimized the number of `sweeps`,[7] the `schedule`[8] and the initial state generator. We noticed, however, that hyperparameters, we found for SA produced worse results compared to the default ones in our following analysis, which may be due to the sensitivity of SA to some of them. For this reason, we retain the default hyperparameters of 1000 `sweeps`, a `geometric beta schedule`, and a random initial state generator.

4 Meta-model training and optimization

In this study, we aim to identify which are the characteristics of a problem that impact the effectiveness of QA. We do this by first training a classification model on the dataset we have created in order to predict whether QA would solve that instance well or not based on its features. Since the classifier is trained to predict the outcome of another experiment, it is called a *meta-model*. Once the meta-model is trained, we can use it to probe how important are the various features.

We train the meta-models with Random Forest, AdaBoost, XGBoost and Logistic Regression, using as input data either a specific domain (e.g., LogIsing, EmbIsing, SolSpace) or a specific component set (e.g., Bias, Coupling, Laplacian), which are described in Section 3.3. The target labels are described in Section 3.2 (i.e., Optimal, ϵ-Optimal, h-

[6] For a detailed explanation of all the hyperparameters of D-Wave Quantum Annealers, refer to the documentation of the devices: https://docs.dwavesys.com/docs/latest/c_solver_parameters.html

[7] A sweep consists in flipping a randomly chosen decision variable.

[8] The schedule determines how the temperature of the system decreases over time.

Table 2 Optimal hyperparameters for QA for each problem class

Problem class	Large instances		Small instances	
	t_a [μs]	n_s	t_a [μs]	n_s
Max-Cut	113	190	200	182
4 × 4-Sudoku	–	–	200	182
Max-Clique	157	165	135	234
Community Detection	92	205	113	259
Number Partitioning	157	165	200	182
Maximum Independent Set	113	190	135	234
Minimum Vertex Cover	27	273	200	182
Set Packing	70	224	157	214
Feature Selection	157	165	27	441
Quadratic Knapsack	5	307	135	234

Optimal) and they are binary, according to whether the solver meets that effectiveness condition or not.

The first step is to train the meta-model and optimize its hyperparameters to ensure it is effective in predicting the label. In order to measure the effectiveness of the meta-models, we have to account for the significant class imbalance of the labels toward the negative class, i.e., instances that are not solved well by QA (see Section 5). We use Balanced Accuracy (BA) to evaluate the meta-models because it is robust to class imbalance. Given the *true positives* as TP, the *true negatives* as TN, the number of positive labels in the data as P and the number of negative labels as N, the Balanced Accuracy BA is computed as follows:

$$BA = \frac{1}{2}\left(\frac{TP}{P} + \frac{TN}{N}\right) \tag{11}$$

The training and optimization of the meta-models is performed with 5-fold Nested Cross-Validation. First, we create a 5 folds *testing split* with a training fold and a testing one which we will use to train and evaluate the meta-model. In order to find the optimal hyperparameters for the meta-model, we split each training fold with a further 5 folds split, the *optimization split*. This results in 5 optimization splits for each of the 5 training folds of the testing split and is aimed at preventing the overfitting of the meta-models. The splits are all stratified with respect to the problem class of the instances, to ensure every split has an equal distribution of the problem classes. All meta-models are trained on the same data splits and we perform different splits for the large and small instances. The hyperparameters of the meta-models are optimized according to a Bayesian Search (Victoria and Maragatham 2021; Snoek et al. 2012) exploring 50 configurations, we select those that provide the best Balanced Accuracy on the optimization split.

Once the meta-models have been optimized, we use them to assess which problem characteristic, i.e., feature, is most important. We use Permutation Feature Importance (PFI), which evaluates how the accuracy of a model drops when the values of a certain feature are shuffled. The idea is that the more important a feature is the larger will be the drop when the values of that feature are shuffled. For each feature the process is repeated multiple times and the corresponding importance is given by the mean of the drop in accuracy observed.

5 Results and analysis

In this section, we provide the most relevant insights of our analysis regarding the effectiveness of QA. We have three goals: (*i*) determine hitch classes of problems are more difficult to solve with QA, (*ii*) understand whether it is possible to predict the effectiveness of QA based on the features we have identified; and (*iii*) discover the domains, the component sets and the features that impact the effectiveness of QA. To do so, we describe the results obtained by solving the instances with QA and with the other classical solvers. Then, we describe the results of the validation of the meta-models and of the Permutation Feature Importance performed on their features. We publish online a dataset with all the instances we generated, the features we computed and the samples obtained for each solver.[9]

5.1 Effectiveness of QA for large instances

In this section, we discuss the effectiveness of QA compared to the other classical solvers (SA, TS and SD) on the large problem instances. Table 3 reports the results on each problem class according to the labels we defined in Section 3.2, i.e., whether the best sample found by QA is at least as good

[9] The instances, the dataset with the features, the results of the solvers and an example script to train meta-models are available at this GitHub repository: https://github.com/qcpolimi/QA-MetaLearning

Table 3 Comparison on the percentage of problem instances in which QA is at least as effective as a specific solver (QA-over-SA, QA-over-TS and QA-over-SD) or as all of them combined (QA-over-all)

Problem class	QA-over-all	QA-over-SA	QA-over-TS	QA-over-SD
Max-Cut	75 %	75 %	75 %	76 %
Number Partitioning	30 %	30 %	33 %	30 %
Community Detection	13 %	15 %	24 %	39 %
Minimum Vertex Cover	0 %	77 %	0 %	0 %
Maximum Independent Set	0 %	73 %	0 %	0 %
Set Packing	0 %	32 %	0 %	0 %
Quadratic Knapsack	0 %	2 %	68 %	0 %
Feature Selection	0 %	0 %	0 %	0 %
Maximum Clique	0 %	0 %	0 %	0 %
Average	14 %	37 %	24 %	18 %

as that found by a specific solver (QA-over-SA, QA-over-TS and QA-over-SD) or by all of them combined (QA-over-all).

As a general comment, we can observe that for less than half of the problem classes (4 out of 9) QA solves effectively more instances than at least one classical solver, while for most of the problem classes (7 out of 9) QA is more effective than at least one of the classical solvers for some particular instances. However, if we combine all classical solvers QA is more effective only in three problem classes but mostly to a limited extent. Only for Max-Cut QA shows a consistently high effectiveness. These results confirm that the effectiveness of QA depends on the problem class, as is the case for classical solvers, which is consistent with what was observed in previous studies (Yarkoni et al. 2022; Jiang and Chu 2023; Huang et al. 2023). If we compare QA and classical solvers throughout the problem classes, we can see that QA is very frequently more effective than SA, while it is more effective than TS or SD only on some specific problem classes. As a result, we conclude that comparing QA only with SA, without considering other solvers, is not the best practice to evaluate the effectiveness of QA.

Regarding the characteristics of the problem classes, a first observation we can make is that QA is more effective on problems that do not require penalties to represent constraints: Max-Cut, Community Detection, and Number Partitioning. This suggests that the presence of constraints is a factor that makes a problem more difficult to solve with QA. The reason for this may be due to the type of quadratic terms introduced by the penalties which could open new research directions in whether one could use a different formulation for the same constraint that is more suitable for QA (Mirkarimi et al. 2024). Furthermore, remember that Max-Cut, Maximum Independent Set, and Minimum Vertex Cover share the same Ising coupling matrix J, with the exception of a multiplicative factor, but have a different bias vector b. We can observe how Max-Cut is the only one among them that is solved effectively by QA, suggesting that the bias structure plays an important role as well.

Lastly, a high number of quadratic terms (i.e., a dense coupling matrix J) does not always negatively affect QA. In particular, both Community Detection, and Number Partitioning have a dense coupling matrix but still 13% of Community Detection instances and the 30% of Number Partitioning instances are solved effectively with QA.

5.2 Effectiveness of QA for small instances

In this section, we discuss the effectiveness of both QA and the other classical solvers (SA, TS, and SD) on the small problem instances. For these instances, we can compute the cost associated with all variable assignments and the global optimum. We do this by computing the Hamiltonian of the problem H_p and using its eigenvalues (i.e., its diagonal). We also compute the maximum energy values needed to assess the ϵ-Optimality for the samples of QA.

Table 4 compares the effectiveness of the solvers according to the labels defined in Section 3.2, i.e., if the solver finds the global optimum (Optimal), if the energy of the best sample is close to that of the global optimum (for ϵ-Optimal, we use $\epsilon = 10^{-5}$), if the variable assignment of the best sample has a Hamming Distance of at most 1 with any of the global optimum solutions (h-Optimal).

Consistently with what was observed for the large instances, QA is less effective than the classical solvers on all the effectiveness conditions. If we compare 10^{-5}-Optimal and

Table 4 Fraction of the instances that are solved well according to a certain effectiveness condition (see Section 3.2)

Solver	Optimal	10^{-5}-Optimal	h-Optimal
QA	43 %	57 %	64 %
SA	59 %	75 %	68 %
TS	74 %	90 %	81 %
SD	**78 %**	**95 %**	**86 %**

The most effective solver is highlighted in bold

h-Optimal, we can see that QA finds more h-Optimal samples than 10^{-5}-Optimal samples, as opposed to the other solvers. This indicates that QA finds more easily samples which are close to the optimal ones in terms of Hamming distance rather than energy. Notice that in this experiment SD is the most effective solver, this may be related to the small size of the instances which may make them relatively easy to solve with simple strategies.

As done for the large instances, we compare the effectiveness of the solvers on the problem classes. Table 5 shows the fraction of problem instances in which the solver finds the global optimum (i.e., Optimal). Overall, as opposed to what we observed for the large instances, on the small ones QA is never more effective than the classical solvers. QA seems to be more effective for problems that are defined over graphs (Max-Cut, Maximum Clique, Community Detection) compared to the ones that are not. Note, however, that on two graph problems, Minimum Vertex Cover and Maximum Independent Set, QA performs significantly behind the classical solvers. Based on the analysis of the large instances (see Section 5.1) we observed that the bias of the problem seems to play a role in affecting the effectiveness of QA. This observation is confirmed here as well since QA is much more effective in solving Max-Cut problems than in solving Maximum Independent Set and Minimum Vertex Cover ones. We can also observe that the effectiveness of QA is quite poor for Set Packing and Feature Selection, being significantly behind the classical solvers.

Interestingly, on Max-Cut and 4 × 4-Sudoku problems almost all the solvers find the global optimum, while for the Quadratic Knapsack hardly any instance can be solved optimally at all, with the most effective solver being TS with 4% of the instances solved optimally.

As a second analysis, we study the effectiveness of QA in sampling solutions with a variable assignment that is close to one of the optimal ones in terms of Hamming distance (i.e., h-Optimality). The results reported in Table 6 are consistent with the previous ones in which we assessed the ability of the solvers to find the global optimum, see Table 5. We should note QA exhibits much better effectiveness, when measured in this way, being able to sample solutions close to the optimal ones in the majority of cases. For example, the effectiveness of the Number Partitioning problem goes up from 33 to 96% while for Minimum Vertex Cover goes from 17 to 58%. The results also confirm that QA is quite effective for problems that do not require penalties to model constraints (Max-Cut, Community Detection, and Number Partitioning). On the other hand, QA is still ineffective for the Feature Selection problem. Quadratic Knapsack remains very challenging for all the solvers.

5.3 Meta-models and feature importance analysis

The goal of our analysis is to identify the domains and the component sets whose features allow the meta-models to predict well the effectiveness of QA. We limit our analysis to the meta-models trained to predict whether QA will be at least as effective compared to all the classical solvers combined (target label QA-over-all) for the large instances. We also analyze the meta-models predicting whether QA will find the global optimum on the small instances (QA-Optimal). The full results are available in the online appendix.

The first important question is whether it is possible to train meta-models able to predict the effectiveness of QA. The results in terms of Balanced Accuracy are shown in Fig. 2 (target QA-over-all) and in Fig. 3 (target QA-Optimal) for the two most effective classifiers. We can immediately see that for several domains or components of the large instances, the Balanced Accuracy is approximately 85%, while for the small instances it is often exceeding 90 or even 95%. This, combined with the fact that we selected a heterogeneous set

Table 5 Fraction of the instances in which the solvers are able to find the global optimum (i.e., Optimal)

Problem class	QA	SA	TS	SD
Max-Cut	**100%**	**100%**	**100%**	88%
Sudoku	**100%**	**100%**	**100%**	**100%**
Maximum Clique	83%	**100%**	92%	**100%**
Community Detection	54%	**58%**	**58%**	**58%**
Number Partitioning	33%	**100%**	92%	**100%**
Maximum Independent Set	21%	17%	**100%**	**100%**
Minimum Vertex Cover	17%	17%	**100%**	96%
Set Packing	12%	50%	54%	**100%**
Feature Selection	0%	**42%**	33%	33%
Quadratic Knapsack	0%	0%	**4%**	0%

The most effective solver of each problem class is highlighted in bold

Table 6 Fraction of the instances in which the solvers are able to find a variable assignment having a Hamming distance of at most 1 with respect to any optimal solution (i.e., h-Optimal)

Problem class	QA	SA	TS	SD
Sudoku	**100%**	**100%**	**100%**	**100%**
Max-Cut	**100%**	**100%**	**100%**	88%
Number Partitioning	96%	**100%**	92%	**100%**
Community Detection	96%	**100%**	96%	**100%**
Maximum Clique	92%	**100%**	92%	**100%**
Minimum Vertex Cover	58%	33%	**100%**	96%
Maximum Independent Set	46%	17%	**100%**	**100%**
Set Packing	38%	50%	54%	**100%**
Feature Selection	4%	71%	67%	**75%**
Quadratic Knapsack	0%	0%	**4%**	0%

The most effective solver of each problem class is highlighted in bold

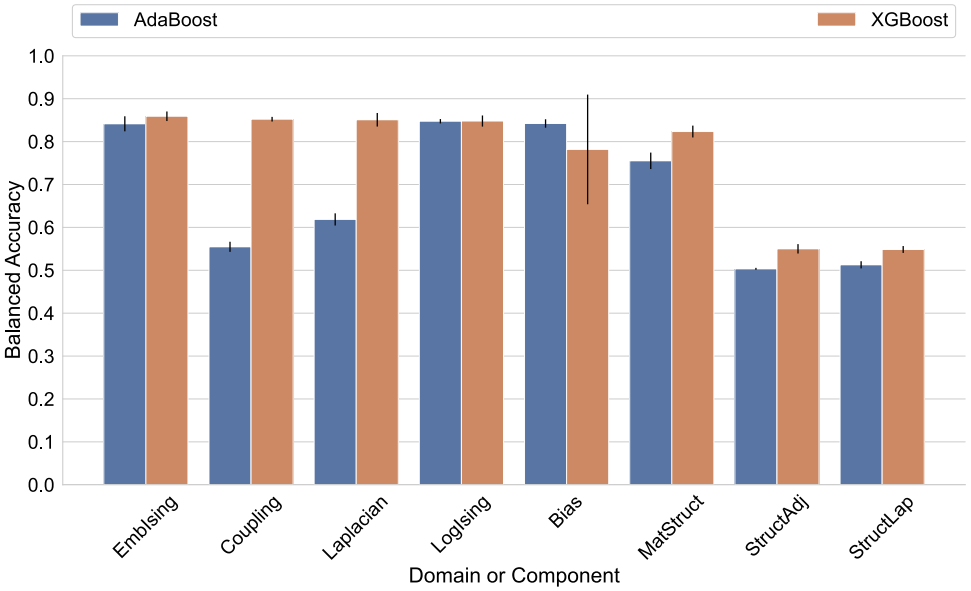


Fig. 2 Bar plot showing the balanced accuracy of the meta-models which predict whether QA is at least as good of all the classical solvers combined (QA-over-all) on the large instances. The domains or component sets the model is trained on are listed on the x-axis. Domains and component sets are ordered according to the Balanced Accuracy of the best related meta-model, in descending order. The vertical black segments on the top of each bar represent the standard deviation of the meta-models

of problem classes and instances that are solved by QA with different degrees of success, allows us to conclude that it is indeed possible to build accurate meta-models to predict the effectiveness of QA. These meta-models can then be used for many purposes, among which, studying the behavior of this technology.

We now analyze which are the domains or components that produce the best meta-models. Concerning the domains, the more informative ones are those related to the graph structure of an Ising problem (LogIsing and EmbIsing) which, based on the high accuracy of the meta-model, we conclude are very informative on the effectiveness of QA. Among the domains, the distribution of the values in the Q matrix of the QUBO problem (MatStruct) is less informative, this can be explained by the fact that the problem that is actually solved on the quantum device is represented as Ising and not as QUBO.

Secondly, if we consider the domains related to the distribution of the energies of an Ising problem which are only available for the small instances (SolSpace, NorMul, 25%-SolSpace, 25%-NorMul in Fig. 3) it is possible to build meta-models which, again, predict well the effectiveness of QA with a Balanced Accuracy well above 90%. This result shows that the effectiveness of QA also depends on the distribution of the energies of the problem, i.e., on how the cost y of the QUBO problem is distributed. Notably, using features based on the solution space allows to achieve comparable Balanced Accuracy with other domains, indicating that both are equally very informative. This is a particularly good

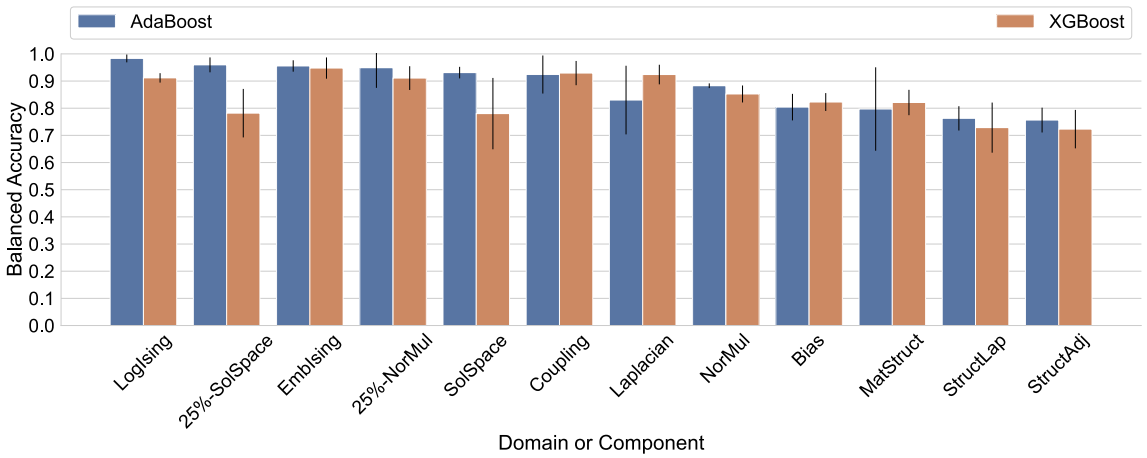


Fig. 3 Bar plot showing the Balanced Accuracy of the meta-models which predict whether QA will find the global optimum (Optimal) on small problem instances. The domains or components the model is trained on are listed on the x-axis. Domains and component sets are ordered according to the Balanced Accuracy of the best related meta-model, in descending order. The vertical black segments on the top of each bar represent the standard deviation of the meta-models

result because it is relatively easy to compute the features for the other domains once the problem has been formulated as QUBO.

If we look at the orthogonal grouping of the features, by component sets, we notice that with the Bias, Coupling the Laplacian component sets it is possible to train at least a meta-model with good Balanced Accuracy (higher than 80%). On the other hand, the Structural Adjacency (StructAdj) and Structural Laplacian (StructLap) are the least informative and, in particular for large instances, do not allow to build a meta-model better than random guess. Since both StructAdj and StructLap are computed on the binarized problem structure, they only account for how the problem variables are connected and not the coefficient values, this means that the structure of the problem alone is not informative at all. The bias and the coupling of an Ising problem, together with the Laplacian matrix related to the graph of the Ising problem, are the most informative on the effectiveness of QA.

In general, we can confirm that the characteristics of the problem are important to determine the effectiveness of QA but one must account for the actual coefficient values of the problem and, preferably, use features derived from the Ising formulation. This could open new research questions on whether one can change the formulation of a problem so that its coefficients have a different distribution that is more adequate for the QA. Furthermore, since the coefficients are a function of the problem class and the data, it may be possible to identify which types of graph topologies may be more or less difficult to tackle based on the distributions of the coefficients that they would produce.

We now move to analyzing which specific features are the most important among the ones we identified. We limit our analysis on feature importance to the XGBoost and AdaBoost meta-models trained on two domains (LogIsing and EmbIsing) and on two component sets (Bias and Coupling). In the case of meta-models related to small instances, we include in the analysis also the domain SolSpace. The best five features of each of these domains and component sets are listed in Table 7 (target QA-over-all, large instances) and in Table 8 (target Optimal, small instances).

Domains feature importance

We consider in particular the domains LogIsing and EmbIsing. The majority of the most important features are related to the bias and to the coupling of the problem, which is consistent with our previous analysis. Some features are related to the distribution of the values of the bias and the coupling (Gini index, Shannon entropy, Herfindahl-Hirschman index), while other features are related to precise values of these mathematical objects (minimum value, maximum value).

Table 7 Best five features, ordered according to feature importance, of AdaBoost and XGBoost meta-models trained with LogIsing and EmbIsing domains and with Bias and Coupling component sets

		AdaBoost	XGBoost
Domains	LogIsing	`Bias min`	`Coupling min eigval`
		`Bias gini index`	`Coupling max eigval`
		`Laplacian connected components`	`Degree min eigval`
		`Laplacian min eigval`	`Coupling radius`
		`Bias condition number`	`Bias gini index`
	EmbIsing	`Bias min`	`Bias hhi`
		`Bias condition number`	`Graph Structure qubits`
		`Bias shannon entropy`	`Coupling spectral gap`
		`Bias gini index`	`Coupling max eigval`
		`Coupling gini index`	`Laplacian spectral gap`
Component sets	Bias	`EmbIsing min`	`LogIsing hhi`
		`LogIsing shannon entropy`	`EmbIsing gini index`
		`LogIsing gini index`	`LogIsing gini index`
		`LogIsing condition number`	`LogIsing max`
		`LogIsing min`	`LogIsing shannon entropy`
	Coupling	`LogIsing gini index`	`LogIsing max eigval`
		`LogIsing hhi`	`LogIsing radius`
		`EmbIsing gini index`	`EmbIsing radius`
		`LogIsing min eigval`	`EmbIsing min eigval`
		`EmbIsing spectral gap`	`LogIsing min eigval`

The target of the meta-models is QA-over-all

Table 8 Best five features, ordered according to feature importance, of AdaBoost and XGBoost meta-models trained on given domains and component sets

		AdaBoost	XGBoost
Domains	LogIsing	`Coupling gini index`	`Coupling max eigval`
		`Bias gini index`	`Degree max eigval`
		`Bias condition number`	`Bias gini index`
		`Coupling radius`	`Bias condition number`
		`Structural Adjacency min eigval`	`Laplacian max eigval`
	EmbIsing	`Bias min`	`Coupling max eigval`
		`Bias shannon entropy`	`Bias gini index`
		`Bias gini index`	`Bias hhi`
		`Degree condition number`	`Structural Degree shannon entropy`
		`Bias hhi`	`Coupling spectral gap`
	SolSpace	`grouped hhi`	`gini index`
		`gini index`	`grouped hhi`
		`third quartile`	`third quartile`
		`shannon entropy`	`hhi`
		`min`	`min`
Component sets	Bias	`EmbIsing condition number`	`EmbIsing gini index`
		`LogIsing hhi`	`EmbIsing condition number`
		`LogIsing shannon entropy`	`LogIsing hhi`
		`LogIsing min`	`EmbIsing shannon entropy`
		`EmbIsing max`	`EmbIsing hhi`
	Coupling	`EmbIsing spectral gap`	`LogIsing spectral gap`
		`LogIsing gini index`	`EmbIsing spectral gap`
		`LogIsing spectral gap`	`LogIsing radius`
		`LogIsing diameter`	`EmbIsing min eigval`
		`EmbIsing min eigval`	`EmbIsing gini index`

The target of the meta-models is QA-Optimal

In particular, notice that `Bias gini index` (related to the distribution of the bias), `Bias condition number` (related to the values of the bias) and `Coupling max eigval` (related to the eigenvalues of the coupling) are among the best features in the majority of the meta-models. We deduce that the distribution of the values and the values themselves of the bias are important to study the effectiveness of QA, together with the eigenvalues of the coupling. This is again an interesting observation because it would allow us to identify in advance whether a problem could be well-suited for QA.

Notice also that the number of qubits needed to embed the problem on the Quantum Annealer (`Graph Structure qubits`) is important, for one meta-model, to predict the effectiveness of QA for the large instances, but not for the small instances. This difference may be linked to the fact that, for the small instances, the number of qubits required after the minor-embedding process is limited and therefore has a lower impact.

We analyze, for these two domains, the least important features too. The majority of them is related to the structural adjacency and to the structural Laplacian matrix. This confirms that the sole structure of a problem is not sufficient to determine the effectiveness of QA. Thus, we must consider also the coefficient values between the variables.

For what concerns the SolSpace domain (see small instances in Table 8), notice that both the meta-models have the same top three features, although in a different order: such features are mostly related to the distribution of the eigenvalues of the problem (`gini index` and `grouped hhi`), which plays, therefore, a role in determining the effectiveness of QA.

Component sets feature importance

If we consider the Bias component set, the majority of the most important features are related to the distribution of the values of the bias, both considered in the LogIsing domain and in the EmbIsing domain. In particular, observe that the

Gini index and the condition number of the Bias, computed in both domains, are among the five most important features, as they were for the meta-models trained on the domains LogIsing and EmbIsing. This corroborate our statement that the distribution and the values of the bias are important to determine the effectiveness of QA.

If we instead consider the Coupling component set, we observe that most of the important features we identify are related to the values and to the eigenvalues of the coupling. Features computed in both the LogIsing and EmbIsing domains are important and the most important features are related to the values of the eigenvalues of the coupling. Notice that the spectral gap and the Gini index of the coupling, computed both in the LogIsing and EmbIsing domains, are shared with almost all meta-models as some of the most important features. We conclude that the eigenvalues of the coupling and their distribution are important to analyze the effectiveness of QA.

Feature correlation with target label

In the previous analysis, we identified the features that are the most important for the meta-models. Some of these features, furthermore, are important also if computed in different domains and for different meta-models. These features are, for the LogIsing and EmbIsing domains: `Bias gini index`, `Bias condition number`, `Coupling max eigval`; while for the SolSpace domain: `gini index`, `grouped hhi`, and `third quartile`. We want now to give an intuition of which values of these features determine a low or an high effectiveness of QA. For each feature we identified, we compute the Spearman rank with the targets of the meta-models (QA-over-all and QA-Optimal).

For most of these features, the Spearman rank does not highlight a strong correlation with the value of the target (in general, the Spearman rank in absolute value is below 0.40). This suggests that the complexity of the underlying behavior might require more powerful tools. Two exceptions are given by `gini index` and `grouped hhi` in domain SolSpace, which have respectively a Spearman rank of -0.596 and 0.537. This mean that as the Gini index of the eigenvalues increases, the Ising problem becomes less difficult to solve. On the other hand, high values of `grouped hhi` of the eigenvalues imply that the Ising problem is difficult to solve with QA.

6 Conclusions

In this paper, we have studied the effectiveness of QA with an empirical approach based on meta-learning models.

First, we select a pool of ten optimization problems which can be formulated as QUBO. Then, we generated more than five thousand instances, based on different problem sizes and structures. In particular, we created two sets, one containing large instances and another with small ones for which we can study also the properties of the whole solution space.

As a second step, we define a set of more than a hundred features to describe each problem instance. The features are heterogeneous, based on graph theory or on metrics largely used in statistics, probability theory, and economics, and account for the structure of the problem, its coefficients, and its solution space. We gather all the features into a meta-learning dataset, which we share on GitHub for further research.

Third, we compare the effectiveness of QA and three classical solvers: Simulated Annealing (SA), Tabu Search (TS), and Steepest Descent (SD). We observe that QA is frequently less effective than the classical solvers, for both the large and small instances, except for specific problems. In particular, we have observed that QA solves more effectively problems with no constraints in their formulation (Max-Cut, Number Partitioning, and Community Detection).

Lastly, we train different classification algorithms to predict whether QA will solve an instance effectively or not and show that it is possible to do so accurately. We then use the meta-models to probe the behavior of QA. In particular, by analyzing the feature importance of the meta-models, we can observe how the distribution of the bias and the coupling of a problem play a key role in determining whether QA will be effective in solving it.

In conclusion, we successfully applied an empirical analysis of the effectiveness of QA based on meta-learning. Possible future directions include the analysis on how different distributions of the coupling and bias values relate to the effectiveness of Quantum Annealing. Such results could be correlated to specific kinds of problems. For example, problems characterized by graphs with a power-law distribution (e.g., problems involving social networks) may be more or less difficult to tackle than those characterized by regular graphs. This can be done, for example, by defining new features that describe how much the distribution of the bias and the coupling differs from another distribution, e.g., from a Gaussian or a uniform distribution. Thanks to its generality, the methodology can be easily extended to other heuristic solvers of QUBO problems, such as the Variational Quantum Eigensolver (VQE) (Fedorov et al. 2022) and Quantum Approximate Optimization Algorithm (QAOA) (Farhi et al. 2014), providing a useful tool to further our understanding of how to use these quantum algorithms effectively.

Supplementary information. The meta-learning dataset with all the problem instances, the corresponding graphs and features, as well the samples obtained with each solver can be accessed here: https://github.com/qcpolimi/QA-MetaLearning

Appendix 1. QUBO formulations of optimization problems

In this Appendix, we show the QUBO formulations of the ten optimization problems we selected for our study. We divide the problems in two different groups: problems defined over a graph, explained in Section 1.1, and problems not defined over a graph, explained in Section 1.2.

1.1 Graph problems

1.1.1 Max-Cut

Given the graph $G = (V, E)$, a cut over graph G induced by the set of vertices S and $V - S$ is the set of edges which connect vertices in S with vertices in $V - S$. The Max-Cut problem aims at finding the largest cut which can be induced on graph G (Glover et al. 2022). Consider, for each vertex i of graph G, a binary variable x_i, such that

$$x_i = \begin{cases} 1 \text{ if } i \in S \\ 0 \text{ otherwise} \end{cases}$$

Then, the Max-Cut problem is formulated as written in Eq. A1

$$\min_x y = -\sum_{(i,j)\in E} (x_i + x_j - 2x_i x_j) \tag{A1}$$

1.1.2 Maximum Independent Set

Given the graph $G = (V, E)$, an independent set is a set S of vertices which are not adjacent to each other. The Maximum Independent Set problem aims at finding the largest independent set in G. Consider, for each vertex i of graph G, a binary variable x_i, such that

$$x_i = \begin{cases} 1 \text{ if } i \in S \\ 0 \text{ otherwise} \end{cases}$$

Then, the Maximum Independent Set problem can be formulated as a QUBO problem as it is written in Eq. A2 (Chapuis et al. 2019).

$$\min_x y = -a \sum_{i\in V} x_i + b \sum_{(i,j)\in E} x_i x_j \tag{A2}$$

The penalty term b must outweigh the term a, to penalize the choice of having two connected vertices in S.

1.1.3 Minimum Vertex Cover

Given the graph $G = (V, E)$, a Vertex Cover C is a set of vertices such that all the edges E are connected at least to a vertex in C. The Minimum Vertex Cover is the Vertex Cover with the smallest number of vertices (Glover et al. 2022). Consider, for each vertex i of graph G, a binary variable x_i, such that

$$x_i = \begin{cases} 1 \text{ if } i \in C \\ 0 \text{ otherwise} \end{cases}$$

The Minimum Vertex Cover can be formulated as a QUBO problem as written in Eq. A3.

$$\min_x y = \sum_{i\in V} x_i + p \cdot \sum_{(i,j)\in E} (1 - x_i - x_j + x_i x_j) \tag{A3}$$

As for the Maximum Independent Set problem, also in the Minimum Vertex Cover problem the penalty term p must be chosen large enough, to penalize the selection of a set of vertices which is not a vertex cover of G.

1.1.4 Max-Clique

Given the graph $G = (V, E)$, a clique C is a sub-graph of G such that C is fully connected. The Max-Clique problem aims at finding the clique C of G with the highest number of nodes. The QUBO formulation of the Max-Clique problem has the same QUBO formulation of the Maximum Independent Set, but it leverages the complement graph $\bar{G}$ of G (Chapuis et al. 2019). Please refer to the Section A.1.2.

1.1.5 Community Detection

Given the graph $G = (V, E)$, the aim of the Community Detection problem is to partition G in two communities C_1, C_2 of similar nodes (Negre et al. 2020; Nembrini et al. 2022). Remember that the graph G is represented by the adjacency matrix A, where the (i, j) entry is $A_{ij} = 1$ if nodes i and j are connected by an edge, otherwise it is A_{ij}. The degree vector d keeps track of how edges are incident to every node of the graph. The similarity between nodes is expressed by the *modularity* matrix B, which is computed according to the following formula:

$$B = A - \frac{dd^T}{2|E|}$$

The binary variable x_i is related to the node $i \in V$ of graph G and it is defined as follows:

$$x_i = \begin{cases} 1 & \text{if } i \in C_1 \\ 0 & \text{if } i \in C_2 \end{cases}$$

The matrix Q is proportional to B. In particular, it is equal to:

$$Q = -\frac{1}{|V|}B$$

The negative sign is due to the fact that QUBO problems require to be *minimization problems* in order to be solved with a QA. The formulation of the problem is therefore given by:

$$\min_x y = -\frac{1}{|V|}x^T Bx$$

1.2 No-graph problems

1.2.1 Number Partitioning

Given a set of real numbers $Z = \{z_1, z_2, ..., z_n\}$, the Number Partition problem aims at finding two partitions Z_1, Z_2 of Z in order to minimize the following expression (Glover et al. 2022):

$$\left(\sum_{z' \in Z_1} z' - \sum_{z'' \in Z_2} z''\right)^2 \tag{A4}$$

The binary decision variable x_i is defined as follows:

$$x_i = \begin{cases} 1 & \text{if } z_i \in Z_1 \\ 0 & \text{if } z_i \in Z_2 \end{cases}$$

The QUBO formulation of the Number Partitioning problem is then derived from the expression (A4), and it can be written as follows:

$$\min_x y = \left(\sum_{i=1}^{|Z|} x_i z_i - \sum_{i=1}^{|Z|}(1 - x_i)z_i\right)^2$$

1.2.2 Set Packing

Given a collection of n sets $S_1, S_2, ..., S_n$, each one with a given capacity $c_1, c_2, ..., c_n$, the Set Packing problem aims at finding a selection of sets providing the largest total capacity, while respecting m constraints for the selection of the sets. The binary decision variable x_i is associated with the set S_i, in particular:

$$x_i = \begin{cases} 1 & \text{if } S_i \text{ is selected} \\ 0 & \text{otherwise} \end{cases}$$

The Set Packing problem can be formulated as it follows:

$$\max_x y = \sum c_i x_i$$
$$\textbf{s.t.} \sum a_{ji} x_i \leq 1, \quad j = 1, 2, ..., m, \quad a_{ji} \in \{0, 1\}$$

In order to map the constraints of the Set Packing problem into a quadratic penalty term, we leverage the following conversion rule (Glover et al. 2022):

$$\sum_{i=1}^{n} a_{ji} x_i \leq 1 \rightarrow p \cdot \sum_{i=1} \sum_{k>i} a_{ji} a_{jk} x_i x_k$$

where coefficients a_{ji} and a_{jk} are either 0 or 1. The objective function is therefore derived as

$$\min_x y = -\sum c_i x_i + p \cdot \sum_{i=1}^{n} \sum_{k>i}^{n} x_i x_k \sum_{j=1}^{m} a_{ji} a_{jk}$$

1.2.3 Quadratic Knapsack

Given a set of projects $P_1, P_2, ..., P_n$ such that for each pair (P_i, P_j) it exists a joint revenue $r_{ij} = r_{ji}$, the Quadratic Knapsack problem aims at maximizing the total revenue of activating the projects under a budget constraint. The Quadratic Knapsack problem is formulated as it follows:

$$\max_x y = \sum_{i}^{n} \sum_{j}^{n} r_{ij} x_i x_j$$
$$\textbf{s.t.} \sum_{i}^{n} c_i x_i \leq b$$

To formulate the Quadratic Knapsack problem in the QUBO formulation, we introduce m binary slack variables $t_1, t_2, ..., t_m$. Each slack variable has a budget coefficient c_{t_j}, which has to be chosen accordingly to the budget b, e.g., by stating that $\sum_j^m c_{t_j} = b$. With these type of constraints, we rewrite the problem using the formulation given in Eq. 2, defined in Section 2.1. The QUBO formulation of the Quadratic Knapsack problem is therefore:

$$\min_x y = -\sum_{i}^{n} \sum_{j}^{n} r_{ij} x_i x_j + p \cdot \left(\sum_{i}^{n} c_i x_i + \sum_{j}^{m} c_{t_j} t_j - b\right)^2$$

1.2.4 Sudoku

A Sudoku can be considered as a constraint satisfaction problem (Bukhari et al. 2022). In our study, we analyze only small instances of Sudoku problems, so we consider Sudoku with a 4×4 grid and some fixed assignments. We call such problems 4×4-Sudoku The binary variable we use is defined as

$$x_{(i,j),k} = \begin{cases} 1 \text{ if cell } (i,\ j) \text{ has value } k \\ 0 \text{ otherwise} \end{cases}$$

There are four kinds of constraints in a Sudoku:

Cell Constraints.
A cell can contain only one number;

$$\sum_{k=1}^{4} x_{(i,j),k} = 1 \quad \forall i,\ j \in \{1, 2, 3, 4\}$$

Row Constraints.
Two cells in the same row must have distinct numbers;

$$\sum_{j=1}^{4} x_{(i,j),k} = 1 \quad \forall i, k \in \{1, 2, 3, 4\}$$

Column Constraints.
Two cells in the same column must have distinct numbers;

$$\sum_{i=1}^{4} x_{(i,j),k} = 1 \quad \forall j, k \in \{1, 2, 3, 4\}$$

Block Constraints.
Two cells in the same block must have distinct numbers;

$$\sum_{(i,j)\in B} x_{(i,j),k} = 1 \quad \forall k \in \{1, 2, 3, 4\}, \quad \forall \text{ blocks B}$$

There are totally 64 constraints in a 4×4-Sudoku problem. If some cells are already assigned, the number of constraints and the number of variables reduces. In particular:

- If cell $(i,\ j)$ is assigned, there exist no variable related to this cell;
- If cell $(i,\ j)$ is not assigned and the number of values it can have is m, then there exist m variables related to cell $(i,\ j)$ and each variable refer to a possible value of the cell;

We can map all the constraints into a single square matrix A, which has at most 64 rows and 64 columns. Each column is related to a variable $x_{(i,j),k}$, while each row refers to a constraint. Assume that x is a vector where each element is a variable $x_{(i,j),k}$. If $\vec{1}$ is a vector of 64 elements, all equal to 1, the QUBO formulation of the 4×4-Sudoku problem is:

$$\min_x y = (Ax - \vec{1})^T (Ax - \vec{1})$$

1.2.5 Feature Selection

Given a dataset $\mathcal{D}$, the Feature Selection problem aims at finding the subset $S \subset F = \{f_1, f_2, ..., f_n\}$ of the best k features able to represent the dataset. This is done especially in machine learning, to reduce the number of features of a predictive model and therefore its complexity. Assume to have n features. For each feature f_i, we have the binary decision variable x_i, defined as follows:

$$x_i = \begin{cases} 1 & \text{if } f_i \text{ is selected} \\ 0 & \text{otherwise} \end{cases}$$

We model the Feature Selection problem using Pearson correlation $Corr(\cdot, \cdot)$ (Ferrari Dacrema et al. 2022). For the quadratic terms, we compute the correlation between two different features f_i, f_j of $\mathcal{D}$. For the linear terms, we compute the correlation between a feature f_i and the target t of $\mathcal{D}$. We can compute directly the QUBO matrix Q, where in particular its $(i,\ j)$ element of the matrix Q is equal to:

$$Q_{ij} = \begin{cases} Corr(f_i, f_j) & \text{if } i \neq j \\ -Corr(f_i, t) & \text{otherwise} \end{cases}$$

Suppose we want to select k features. To do so, we consider the following QUBO formulation of the Feature Selection problem:

$$\min_x y = x^T Q x + \left(\sum_{i=1}^{n} x_i - k \right)^2$$

Notice that, if less or more than k features are selected, the penalty term is larger than 0 and the objective function value gets worse.

Appendix 2. Instance generation strategies

In this section, we explain the strategies we applied to generate the instances of the optimization problems we selected. The strategies vary whether the problem is defined over a graph or not. An instance of a problem is characterized by

its *structure*, which is either the topology of a graph or a general title which refers to the strategy used to generate the constraints and the objective function, and the number of variables n.

In general, for all the problems we generate instances with a minimum of $n_{\min}$ variables to a maximum of $n_{\max}$ variables. For every optimization problem, except for the 4×4-Sudoku and for the Feature Selection problem, there exist n_{rep} instances having a certain structure and with the same number of variables. Such instances are however different between each other, thanks to graph tweaking (explained in Section B.1) and to the random generation of the coefficient of the cost function and of the constraints.

For the small instances, we have chosen $n_{\min} = 27$, $n_{\max} = 32$ and $n_{rep} = 1$. For the large instances, we have chosen $n_{\min} = 69$, $n_{\max} = 99$ and $n_{rep} = 5$.

2.1 Graph problems

The instance of a problem defined over a graph is totally determined by the topology of the graph. The strategy we applied to generate instances of graph problems is essentially divided in two distinct phases: the first is the generation of the graph G of n nodes, according to a certain topology; the second is the generation of an instance, for every graph problem, defined over G.

We have selected four different graph topologies to generate the graph G: the Star topology, the Cycle topology, the 2d-grid topology and the Erdös-Rényi topology. A key step in the generation of G is the random insertion and removal of a limited number of edges, resulting in the *tweaking* of graph G. In this way, we generate more graphs starting from a given topology and with n nodes, which however get tweaked in different ways. We ensure always that the tweaked graph is connected. In particular, the graph tweaking process happens according to the following rules:

- At most $n_{ins} = \lfloor \frac{n}{6} \rfloor$ edges are inserted;
- At most $n_{rem} = \lfloor \frac{n}{8} \rfloor$ edges are removed;
- To perform the tweaking, we randomly modify the adjacency matrix A of the original graph. We start the tweaking of A from the first element of the first row, A_{11}, and we proceed left-to-right, till the element in the last column and in the last row, A_{nn}.
- If $A_{ij} = 0$ and less than n_{ins} edges have been inserted, insert the edge (i, j) by setting $A_{ij} = 1$ with probability $p_{ins} = 40\%$;
- On the contrary, if $A_{ij} = 1$ and less than n_{rem} edges have been removed, remove the edge (i, j) by setting $A_{ij} = 0$ with probability $p_{rem} = 30\%$;
- If the tweaked graph is not connected, repeat the procedure;

For the Minimum Vertex Cover and the Maximum Independent Set problems we have to set the value of the penalty term coefficient p. For the Minimum Vertex Cover, we have set $p = n$, that is to the number of nodes of the graph G. For the Maximum Independent Set, we have set $p = 2n$. In both cases, we have that an assignment which violates a constraint highly penalize the objective function and that it is not a solution of the instance.

2.2 No-graph problems

For what concerns the problems not defined over a graph, to generate an instance we have to generate the matrices related to the objective function and to the constraints. In particular, we want to analyze satisfiable instances. We discuss how the strategies we used to generate the instances of each problem we selected.

2.2.1 Number Partitioning

To generate the instances of the Number Partitioning problem, we have to define the set Z to partition. For simplicity, we generate only sets of integer numbers. We selected three probability distributions to generate the set Z, in addition to a fourth strategy where Z contains all the numbers between 1 and the number of variables n. The three probability distributions are: (*i*) a uniform distribution between 1 and 99; (*ii*) a geometric distribution with probability of success $p = 0.02$; (*iii*) a Poisson distribution of mean value $\mu = 50$.

2.2.2 Set Packing

An instance of the Set Packing problem is determined by the capacities of the n sets $c_1, c_2, ..., c_n$, the coefficients $a_{j,i}$ of the constraints, and the coefficient of the penalty term p.

For what concerns the capacities c_i, we sample the highest possible capacity coefficient, called $c_{\max}$, from a uniform distribution between 10 and 39. This number represents the highest possible capacity coefficient, called $c_{\max}$. Then, we generate the capacities $c_1, c_2, ..., c_n$ of the sets by sampling them from to a uniform distribution between 1 and $c_{\max}$.

To generate the coefficients $a_{j,i}$, we consider a matrix A where the element in the j-th row and in the i-th columns is equal to $a_{j,i}$. Notice that A has always n columns. We have defined four strategies to generate matrix A:

- A is rectangular, with $n - 1$ rows and the elements $a_{i,i}$ and $a_{i,i+1}$ are equal to 1, for $i = 1, 2, ..., n - 1$. All the other elements are set to zero. We say that this matrix has a *step* structure. We provide below, as an example, a

matrix of this kind with n = 5:

$$A = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \end{pmatrix}$$

- A is rectangular, with a number of rows m randomly sampled from a uniform distribution between 2 and $\lfloor \frac{n}{2} \rfloor$. All the columns of A have exactly one element equal to 1. We say that this matrix has *disjoint rows* structure. Below, we provide an example of a disjoint rows matrix, with n = 5 and m = 3:

$$A = \begin{pmatrix} 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \end{pmatrix}$$

- A is a square matrix. The elements on the main diagonal are all equal to 1. Furthermore, for each row of A, a random off-diagonal element is set to 1 with probability $p = 60\%$. We say that this matrix has an *almost diagonal* structure;
- A is rectangular, with a number of rows m sampled from a uniform distribution between 10 and $\left\lfloor \frac{n^2}{2} \right\rfloor$. The elements of A are randomly generated between 0 and 1, with equal probability. We say that this matrix has a *random* structure;

Also the penalty term p must be chosen, in order to penalize the assignments which violate the constraints. We use 100 steps of Bayesian optimization to choose p, by maximizing the percentage of feasible solutions found by Simulated Annealing in 30 executions. The range of values of p is determined by the sum $c_{all} = \sum_i^n |c_i|$. In particular:

$$\left\lfloor \frac{c_{all}}{3} \right\rfloor \leq p \leq c_{all}$$

2.2.3 Quadratic Knapsack

In the Quadratic Knapsack problem, to generate the instances we have to choose: (*i*) the values of the joint revenue r_{ij}; (*ii*) the values of the coefficients $c_1, c_2, ..., c_n$ of the constraint; (*iii*) the budget b; (*iv*) the number m of slack variables t_i to introduce and their constraints coefficients c_{t_i}; (*v*) the value of the penalty term coefficient p.

We generate the joint revenue values by generating a matrix. The general joint revenue value r_{ij} is the element in the i-th row and in the j-th column of a revenue matrix R. By definition of the Quadratic Knapsack problem, R is symmetric. We generate R according to the following four strategies:

- R is diagonal. The elements on the diagonal are sampled from a uniform distribution of integer numbers between 15 and 39. We say that the structure of R is *diagonal*;
- R has the whole diagonal, with some other few random off-diagonal elements. The elements on the diagonal of R are sampled from a uniform distribution of integers numbers between 15 and 39. Then, for every column of R, we set with probability 40% a random off-diagonal r_{ij} with an integer number sampled from a uniform distribution between 1 and 24; when the element is set, we make the matrix symmetric by dividing r_{ij} by 2 and setting $r_{ji} = r_{ij}$. In the case r_{ji} element is set again, when considering column j, the previous value of r_{ji} and r_{ij} is overwritten. We say that R has an *almost diagonal* structure. An example of almost diagonal 4×4 matrix is:

$$\begin{pmatrix} 17.0 & 13.5 & 0.0 & 16.0 \\ 13.5 & 32.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 19.0 & 0.0 \\ 16.0 & 0.0 & 0.0 & 26.0 \end{pmatrix}$$

- The on-diagonal elements of R are non-zero; also the elements immediately on the right and on the left of on-diagonal elements are non-zero; the on-diagonal elements are integer numbers sampled from a uniform distribution between 15 and 39; the off-diagonal elements immediately to the right of on-diagonal elements are sampled from the same distribution; then, the off-diagonal elements are divided by 2 and the matrix is made symmetric, by setting $r_{ij} = r_{ji}$, for every $i, j \in \{1, 2, ..., n\}$. We say that R has an *enlarged diagonal* structure. An example of a 4×4 enlarged diagonal is:

$$\begin{pmatrix} 15.0 & 11.0 & 0.0 & 0.0 \\ 11.0 & 25.0 & 9.5 & 0.0 \\ 0.0 & 9.5 & 31.0 & 17.0 \\ 0.0 & 0.0 & 17.0 & 37.0 \end{pmatrix}$$

- R is generated randomly, and every element is sampled from the uniform distribution of integer numbers between 15 and 39; then, all the elements below the diagonal are ignored and set to 0; finally, the off-diagonal elements are divided by 2 and the matrix is made symmetric, by setting $r_{ij} = r_{ji}$, for every $i, j \in \{1, 2, ..., n\}$. We say that R has a *random* structure.

The coefficients $c_1, c_2, ..., c_n$ which appear in the constraint are integer numbers sampled from a uniform distribution between 1 and 15. The budget b is set as $\phi * \sum_{i=1}^n c_i$, where $\phi \in \mathbb{R}$ is sampled from a uniform distribution between 0.20 and 0.70.

We have chosen to introduce four binary slack variables t_1, t_2, t_3, t_4 to transform the inequality constraints into equality constraints. The slack variables are respectively associated with the budget coefficients $c_{t_1}, c_{t_2}, c_{t_3}, c_{t_4}$. We choose also the values of these coefficients as it follows:

$$c_{t_1} = \left\lfloor \frac{b}{2} \right\rfloor \quad c_{t_2} = \left\lfloor \frac{b}{4} \right\rfloor \quad c_{t_3} = \left\lfloor \frac{b}{8} \right\rfloor \quad c_{t_4} = b - \sum_{i=1}^{3} c_{t_i} \tag{B5}$$

We use 100 steps Bayesian Search to choose p, by maximizing the percentage of feasible solutions found by Simulated Annealing in 30 executions. The range of the values of p depends on the sum $r_{all} = \sum_{i=1}^{n} \sum_{j=1}^{n} |r_{ij}|$. In particular:

$$1 \leq p \leq r_{all}$$

2.2.4 Sudoku

To generate 4×4-Sudoku instances, we generated randomly 30 different solved 4×4-Sudoku games. We then iteratively removed the value of a cell, chosen randomly, of the 4×4-Sudoku game. Each time we remove a cell, we increase the number of variables of the related 4×4-Sudoku instance. We ensure that the generated instance has a number of variables between 27 and 32.

Notice that it is not possible to build instances larger than 64 variables of the 4×4-Sudoku problem. For this reason, we generate only small instances for this kind of problem.

2.2.5 Feature Selection

We generated the instances of the Feature Selection problem starting from the following public datasets[10]: `waveform-5000`, `SPECTF`, `spambase`, `USPS`, `isolet`, `gisette`, `Bioresponse`. An instance of the Feature Selection problem depends on the dataset $\mathcal{D}$, on the number of features to select k, and on the target variable t.

Assume we generate m instances with a number of variables n, with n bounded between $n_{\min}$ and $n_{\max}$. To select the dataset D, we use an iterative procedure. We start from the dataset `waveform-5000` and we check if it has more than $n = n_{\min}$ non-target features: if this condition is true, we select this dataset and we reduce its number of features to n, by deleting randomly chosen features; otherwise, we go to the next dataset, chosen according to the order we used to list them, and we repeat this check. After the last dataset, `Bioresponse`, we repeat staring from `waveform-5000`. After that m features are selected, we continue the procedure by incrementing n. The procedure goes on until we generate m instances with $n = n_{\max}$ variables.

Assume that the dataset $\mathcal{D}$ is the i-th dataset selected to generate an instance of n variables, in the iterative procedure, the number of features to select is computed as follows:

$$k = \left\lfloor \frac{n}{5} \right\rfloor + i$$

For what concerns the target variable t, every dataset we selected has one or multiple target variables. Since this formulation can tackle only one target variable t, in case of multiple target variables we randomly select one of them and delete all the others.

For small instances, we considered $n_{\min} = 27, n_{\max} = 32$ and $m = 4$, for a total of 24 instances. The number of features k we select is bounded between 5 and 9. For large instances, we considered $n_{\min} = 69$, $n_{\max} = 99$ and $m = 5$, for a total of 155 instances. The number of features we select is bounded between 13 and 23.

Appendix 3. Definition of the features

We introduce in this section the definitions of the features we used in our study. A subset of these features is based on probability theory and statistics, another subset is based on graph theory, and other features are related to the study of the spectrum of matrices.

Gini index

Given a set X of positive numbers, the Gini index $Gini(Y)$ is a real number which measures the degree of inequality between the values $y \in Y$ (Damgaard and Weiner 2000). It is comprised between 0 (all the values $y \in Y$ are equal) and 1 (only one value of Y is different from 0). Given an ordered collection of values $Y = \{y_1, y_2, \cdots, y_m\}$, such that $0 \leq y_1 \leq y_2 \leq \cdots \leq y_m$, the Gini index is computed as follows:

$$g(Y) = \frac{2 \sum_{i=1}^{m} i y_i}{m \sum_{i=1}^{m} y_i} - \frac{n+1}{n} \tag{C6}$$

Herdindahl-Hirschman index

Given a set Φ, called *industry*, composed of m firms $F_1, F_2, ..., F_m$, each one characterized by a market share $S(F_i) \in [0, 1]$, such that $\sum_{i=1}^{m} F_i = 1$. The Herfindahl-Hirschman index (HHI) is a measure used in economics to quantify the competitiveness of an industry with respect to the

[10] Datasets are available on the website OpenML: https://www.openml.org

market share of the firms that compose the industry (Brezina et al. 2016). The HHI of industry Φ is defined as follows:

$$HHI(\Phi) = \sum_{F \in \Phi} S(F)^2$$

Clearly, $0 < HHI(\Phi) \leq 1$. In particular, $HHI(\Phi) = 1$ when there exists only one firm inside industry Φ, that is Φ is a monopoly. On the contrary, if we assume that all the firms have equal market share, the market is competitive and, if $m \to +\infty$, we have that $HHI(\Phi) \to 0$.

If two firms F_i, F_j merge into a larger new firm F_z, we have that $S(F_z) = S(F_i) + S(F_j)$. The HHI computed on the new industry, which includes F_z, increases.

Shannon entropy

Given a random variable Y, distributed according to a certain distribution $p_Y(y)$. The Shannon entropy of Y measures the level of uncertainty in the distribution $p_Y(y)$. In the discrete and finite case, assuming that the possible values of Y are $y_1, y_2, ..., y_m$, Shannon entropy is defined as follows:

$$Sh(Y) = -\sum_{i=1}^{m} p_Y(y_i) \log_2 p_Y(y_i)$$

In general, Shannon entropy can be greater than 1. The maximum value of Shannon entropy occurs when Y is distributed according to a uniform distribution, where all the values of Y are equally probable and none of them can be predicted more easily than the others.

Condition number

Given a matrix M, the condition number $CN(M)$ measures how close is matrix M to be singular. If M is symmetric and real, the $CN(M)$ is computed according to the maximal and minimal eigenvalues of M, respectively $\lambda_{\max}$ and $\lambda_{\min}$:

$$CN(M) = \left| \frac{\lambda_{\max}}{\lambda_{\min}} \right|$$

Radius of a graph

Given a graph $G(V, E)$, described by the adjacency matrix A. The *radius* of graph G is the largest eigenvalue of A in absolute value.

Diameter of a graph

Given a graph $G(V, E)$, the shortest path between two nodes i, j is the sequence of edges having minimal cost which connects i and j. The *diameter* of G is the length of the longest shortest path. If G is not connected, the diameter is not defined. In the case of negative weights on the edges that form a cycle, the diameter is not defined too.

Spectral gap of a graph

Given a graph $G(V, E)$, described by the adjacency matrix A, the *spectral gap* of G the difference, in modulus, between the two largest eigenvalues of A. Another definition we use is related to the Laplacian L of the graph. In this case, the spectral gap is the smallest non-zero eigenvalue of the Laplacian L related to the graph.

Connectivity of graph

Given a graph $G(V, E)$, described by a positive semi-definite Laplacian matrix L, the connectivity is equal to the second smallest eigenvalue of L.

Connected components of a graph

Given a graph $G(V, E)$, described by a positive semi-definite Laplacian matrix L, the number of connected components of the graph is equal to the multiplicity of the eigenvalue 0 of L.

Appendix 4. Domains and component sets

4.1 Matrices of an Ising graph

We call the Ising graph a graph having the coupling matrix J of an Ising problem as an adjacency matrix. Each node of the graph corresponds to a variable of the Ising problem. The nodes of an Ising graph have a weight too, determined by the bias vector b of the Ising problem. In the case the Ising problem is mapped onto the topology of a Quantum Annealer, we call it *embedded Ising graph*; otherwise, we refer to it as *logical Ising graph*. We call the structural adjacency matrix A of an Ising graph the adjacency matrix related to the unweighted version of the Ising graph.

The degree matrix D_J of the Ising graph is computed starting from the coupling J, while the structural degree matrix D_A of the Ising graph is computed starting from A.

The Laplacian L of an Ising graph is computed according to the following formula:

$$L = D - J$$

The structural Laplacian L_A is computed using A instead of J and D_A instead of D_J.

The normalized adjacency matrix A_N is computed according the following formula:

$$A_N = D_A^{-\frac{1}{2}} A D_A^{-\frac{1}{2}}$$

The normalized Laplacian L_N is instead equal to:

$$L_N = D_A^{-\frac{1}{2}} L_A D_A^{-\frac{1}{2}}$$

4.2 Domains and related features

We list here all the features computed in every domain we consider. We refer to the definitions of the features given in Appendix C.

4.2.1 Embedding and logical Ising graph (EmbIsing and LogIsing)

In these domain, we gather features related to the embedded and logical Ising graph. We compute the features starting from the following mathematical objects: bias, coupling, Laplacian, degree, structural adjacency, structural degree, structural Laplacian, normalized adjacency and normalized Laplacian matrices. In both the domains, we compute the same features, but the considered mathematical objects vary in size and values. For both the domains, we use the notation `Object feature` to refer to the feature `feature`, in lowercase letters, computed on the mathematical object `Object`.

Coupling
Given the coupling J, we compute:

- `Coupling gini index`: Gini index of the eigenvalues of J. Since eigenvalues may be negative, we shift all the eigenvalues by the subtracting to all of them the smallest eigenvalue of J;
- `Coupling hhi`: HHI of the eigenvalues of J. The share of the eigenvalues is equal to their multiplicity divided by n;
- `Coupling shannon entropy`: Shannon entropy of the eigenvalues of J. The probability of an eigenvalue is equal to its multiplicity divided by n;
- `Coupling condition number`: condition number of J;
- `Coupling radius`: radius of the graph described by adjacency matrix J;
- `Coupling diameter`: diameter of the graph described by the adjacency matrix J. It may be not defined;
- `Coupling spectral gap`: spectral gap of the graph described by the adjacency matrix J;
- `Coupling min eigval`: smallest eigenvalue of J;
- `Coupling max eigval`: largest eigenvalue of J.

Bias
Given the bias b, we compute:

- `Bias gini index`: Gini index of the values of b. Since some values of b may be negative, the values are all shifted by subtracting the minimal value of b;
- `Bias hhi`: HHI of the values of b. The share of the values corresponds to their multiplicity divided by n;
- `Bias shannon entropy`: Shannon entropy of the values of b. The probability of a value corresponds to its multiplicity divided by n;
- `Bias min`: the minimal value of b;
- `Bias max`: the maximal value of b;
- `Bias condition number`: the fraction between the maximal and the minimal value of b, in absolute value.

Degree
Given the degree matrix D of an Ising graph, we compute:

- `Degree gini index`: Gini index of the eigenvalues of D. Since eigenvalues may be negative, we shift all the eigenvalues by the subtracting to all of them the smallest eigenvalue of D;
- `Degree hhi`: HHI of the eigenvalues of D. The share of the eigenvalues is equal to their multiplicity divided by n;
- `Degree shannon entropy`: Shannon entropy of the eigenvalues of D. The probability of an eigenvalue is equal to their multiplicity divided by n;
- `Degree min eigval`: minimal eigenvalue of D;
- `Degree max eigval`: maximal eigenvalue of D;
- `Degree condition number`: condition number of D.

Laplacian
Given the Laplacian matrix L of an Ising graph, we compute:

- `Laplacian gini index`: Gini index of the eigenvalues of L. Since eigenvalues may be negative, we shift all the eigenvalues by the subtracting to all of them the smallest eigenvalue of L;
- `Laplacian hhi`: HHI of the eigenvalues of L. The share of the eigenvalues is equal to their multiplicity divided by n;
- `Laplacian shannon entropy`: Shannon entropy of the eigenvalues of L. The probability of an eigenvalue is equal to its multiplicity divided by n;
- `Laplacian min eigval`: minimal eigenvalue of L;
- `Laplacian max eigval`: maximal eigenvalue of L;
- `Laplacian connectivity`: the second smallest eigenvalue of L. We compute it also if L is not positive semi-definite;
- `Laplacian spectral gap`: the smallest non-zero eigenvalue of L;

- `Laplacian connected components`: multiplicity of eigenvalue 0 of L. We compute it also if L is not positive semi-definite.

Structural and normalized matrices

The features computed for the structural adjacency, degree and Laplacian matrices and for the normalized adjacency and Laplacian matrices are the same computed for their weighted counterparts. The main difference is that the structural and the normalized matrices are positive-semidefinite and there is no need to shift the eigenvalues to compute the Gini index.

Graph structure in EmbIsing domain

In the EmbIsing domain, we compute also two features related to the size of the Embedded Ising graph. In particular, we compute:

- `Graph Structure qubits`: number of nodes of the Embedded Ising graph, which corresponds to the number of qubits used to solve an Ising problem with the Quantum Annealer;
- `Graph Structure chains`: number of chains of nodes in the Embedded Ising graph; a chain represents a single node of the Logical Ising graph, i.e., a single variable of the original Ising problem.

4.2.2 Matrix structure (MatStruct)

In this domain, we compute the features related to the values of the QUBO matrix Q. In particular, we compute:

- `gini index`: Gini index of the values of Q. We subtract the minimal value of Q from all the elements to guarantee that the values are positive;
- `hhi`: HHI of the values of Q. The share of an element of Q is equal to its number of occurrences inside Q, divided by n^2;
- `shannon entropy`: Shannon entropy of the values of Q. The probability of an element of Q is equal to its number of occurrences inside Q, divided by n^2.

4.2.3 Solution space (SolSpace)

In this section, we compute features related to the eigenvalues of the Hamiltonian of the problem H_p, related to a QUBO problem. A generic eigenvalue of H_p is λ_i. Assume that $\lambda_{\min}$ is the minimal eigenvalue of H_p. Remember that, if the QUBO problem has n variable, H_p has 2^n eigenvalues. In this domain, we compute:

- `gini index`: Gini index of the eigenvalues of H_p. To have all positive values, eigenvalues are shifted by subtracting $\lambda_{\min}$ to all of them;
- `hhi`: HHI of the eigenvalues of H_p. We use the value

$$S(\lambda_i) = \frac{\lambda_i - \lambda_{\min}}{\sum_{j=1}^{2^n}(\lambda_j - \lambda_{\min})} \tag{D7}$$

  as the share of an eigenvalue λ_i;
- `grouped hhi`: HHI of the eigenvalues of H_p, where identical eigenvalues are considered together. The share of the eigenvalue λ_i is equal to the sum of the shares of the eigenvalues equal to λ_i, computed according to the formula (D7).
- `shannon entropy`: Shannon entropy of the eigenvalues of H_p. We use as probability of eigenvalue λ_i the value described in Eq. D7;
- `min`: the minimal eigenvalue of H_p;
- `first quartile`: the eigenvalue of the first quartile of H_p;
- `median`: the median eigenvalue of H_p;
- `third quartile`: the eigenvalue of the third quartile of H_p;
- `max`: the maximal eigenvalue of H_p.

4.2.4 Normalized multiplicity (NorMul)

In this section, we compute features related to the multiplicity of the eigenvalues of the Hamiltonian of the problem H_p, related to a QUBO problem. Assume that there are m different eigenvalues of H_p. The multiplicity of the eigenvalue λ_i of H_p is equal to μ_i. We call *normalized multiplicity* π_i of λ_i the value:

$$\pi_i = \frac{\mu_i}{\sum_{j=1}^{m} \mu_j}$$

In this domain, we compute the following features:

- `gini index`: Gini Index of the normalized multiplicities of the eigenvalues of H_p;
- `hhi`: HHI of eigenvalues of H_p, where we use their normalized multiplicities as shares;
- `shannon entropy`: Shannon entropy of the eigenvalues of H_p, where we use their normalized multiplicities as probabilities;
- `smallest eig`: the normalized multiplicity of the smallest eigenvalue of H_p.

4.2.5 25%-SolSpace and 25%-NorMul

The features computed in these domains are the same computed in the domain SolSpace and NorMul. The only difference is that here, if consider the eigenvalues in ascending order, we consider only the eigenvalues of H_p in the first quartile.

4.3 Component sets

Component sets gather all the features computed on the same mathematical objects, but in different domains. In these study, all the component sets contain features computed in both the LogIsing and EmbIsing domains. For every component set, we use the notation `Domain feature` to refer to the feature `feature`, written in lowercase letters, computed in the domain `Domain` on the mathematical object related to the component set. The component sets we consider are:

- **Bias**: it contains the features related to the bias;
- **Coupling**: it contains the features related to the coupling;
- **Laplacian**: it contains the features related to the Laplacian matrix;
- **StructAdj**: it contains the features related to the structural adjacency matrix;
- **StructLap**: it contains the features related to the structural Laplacian matrix;

4.4 Complexity in features computation

The major complexity in the computation of the features is given by the computation of the Hamiltonian of the problem, H_p. To do so it is in fact necessary to compute a 2^n real-valued vector, with n number of QUBO variables, according to Eq. 5. This of course is related only to the small instances.

For what concerns the features related to the large instances, the bottleneck lies in computing all the minor embeddings of the instances: minor embedding is, indeed, an NP-Hard problem.

Once H_p and the minor embedding of the instances have been computed, all the features are easily computed without any particular complexity.

Appendix 5. Hyperparameters of the solvers

Tabu Search

We optimize the number of restarts of the algorithm. The default number of restarts is 100,000. Table 9 contains the optimal hyperparameters found for each problem class.

Table 9 Optimal hyperparameters of Tabu Search for each problem class

Problem class	Number of restarts	
	Large instances	Small instances
Max-Cut	523,486	990,687
4 × 4-Sudoku	–	72,057
Max-Clique	977,117	1,174,131
Community Detection	1,415,919	372,449
Number Partitioning	1,350,888	871,728
Maximum Independent Set	869,152	1,436,896
Minimum Vertex Cover	388,887	1,047,039
Set Packing	890,741	1,445,991
Feature Selection	1,037,380	995,175
Quadratic Knapsack	452,422	1,305,536

Table 10 Optimal hyperparameters for simulated annealing for each problem class

Problem class	Large instances		Small instances	
	`sweeps`	`schedule type`	`sweeps`	`schedule type`
Max-Cut	924	`linear`	592	`linear`
4 × 4-Sudoku	–	–	1260	`geometric`
Max-Clique	630	`geometric`	728	`geometric`
Community Detection	1073	`linear`	1033	`linear`
Number Partitioning	1426	`geometric`	819	`linear`
Maximum Independent Set	626	`geometric`	1192	`linear`
Minimum Vertex Cover	1100	`geometric`	1393	`linear`
Set Packing	625	`geometric`	518	`geometric`
Feature Selection	1388	`linear`	576	`linear`
Quadratic Knapsack	503	`geometric`	1133	`linear`

Simulated Annealing

We optimized the number of `sweeps` and the `schedule type`. The default value of the number of `sweeps` is 1000, while the defualt `schedule type` is `geometric`. Notice that we solve the instances also with Simulated Annealing with no hyperparameters optimization and we obtained better results. Table 10 contains the optimal hyperparameters found for each problem class.

Appendix 6. Additional results on the effectiveness of solvers

Table 11 Table containing, for each considered problems, the fraction of 10^{-5}-optimally solved instances for all the solvers

Solver / Problem	10^{-5}-Optimal QA	SA	TS	SD
Max-Cut	**100%**	**100%**	**100%**	88%
Sudoku	**100%**	**100%**	**100%**	**100%**
Maximum Clique	83%	**100%**	92%	**100%**
Community Detection	**100%**	**100%**	**100%**	**100%**
Number Partitioning	46%	**100%**	92%	**100%**
Maximum Independent Set	21%	17%	**100%**	**100%**
Minimum Vertex Cover	17%	17%	**100%**	96%
Set Packing	25%	50%	58%	**100%**
Feature Selection	0%	71%	**75%**	**75%**
Quadratic Knapsack	62%	88%	79%	**92%**

The values in bold text refer to the most effective solvers for a particular problem

Author Contribution All authors contributed to the study conception and design. M. Ferrari Dacrema conceived the methodology. Material preparation, data collection and analysis were performed by R. Pellini. The first draft of the manuscript was written by R. Pellini and all authors contributed to the final version. All authors read and approved the final manuscript.

Funding Open access funding provided by Politecnico di Milano within the CRUI-CARE Agreement. We acknowledge the financial support from ICSC - "National Research Centre in High Performance Computing, Big Data, and Quantum Computing," funded by European Union - NextGenerationEU. We acknowledge the CINECA award under the ISCRA initiative, for the availability of high-performance computing resources and support. We also acknowledge the support and computational resources provided by E4 Computer Engineering S.p.A.

Data Availability The instances, the dataset with the features, the results of the solvers and an example script to train meta-models are available at this GitHub repository: https://github.com/qcpolimi/QA-MetaLearning.

Declarations

Conflict of Interest The authors declare no competing interests.

References

Albash T, Lidar DA (2018) Adiabatic quantum computation. Rev Mod Phys 90:015002. https://doi.org/10.1103/RevModPhys.90.015002

Boothby K, Bunyk P, Raymond J, Roy A (2020) Next-generation topology of d-wave quantum processors. https://doi.org/10.48550/arXiv.2003.00133

Born M, Fock V (1928) Beweis des adiabatensatzes. Zeitschrift für Physik 51:165–180

Brezina I, Pekár J, Čičková Z, Reiff M (2016) Herfindahl–Hirschman index level of concentration values modification and analysis of their change. CEJOR 24:49–72. https://doi.org/10.1007/s10100-014-0350-y

Bukhari F, Nurdiati S, Najib M, Safiqri N (2022) Formulation of sudoku puzzle using binary integer linear programming and its implementation in Julia, Python, and Minizinc. Jambura J Math 4. https://doi.org/10.34312/jjom.v4i2.14194

Cai J, Macready WG, Roy A (2014) A practical heuristic for finding graph minors. arXiv e-prints arXiv:1406.2741. https://doi.org/10.48550/arXiv.1406.2741

Carmesin J (2022) Graph theory – a survey on the occasion of the Abel prize for lászló lovász. Jahresber Deutsch Math-Verein 124:83–108. https://doi.org/10.1365/s13291-022-00247-7

Chapuis G, Djidjev H, Hahn G, Rizk G (2019) Finding maximum cliques on the d-wave quantum annealer. J Signal Process Syst 91:363–377. https://doi.org/10.1007/s11265-018-1357-8

Choi V (2008) Minor-embedding in adiabatic quantum computation: I. The parameter setting problem. Quantum Inf Process 7:193–209. https://doi.org/10.1007/s11128-008-0082-9

Damgaard C, Weiner J (2000) Describing inequality in plant size or fecundity. Ecology 81:1139–1142. https://doi.org/10.1890/0012-9658(2000)081[1139:DIIPSO]2.0.CO;2

Farhi E, Goldstone J, Gutmann S (2014) A quantum approximate optimization algorithm. http://arxiv.org/abs/1411.4028

Farhi E, Goldstone J, Gutmann S, Sipser M (2000) Quantum computation by adiabatic evolution. https://arxiv.org/abs/quant-ph/0001106

Fedorov DA, Peng B, Govind N, Alexeev Y (2022) VQE method: a short survey and recent developments. Materials Theory 6:2. https://doi.org/10.1186/s41313-021-00032-6

Ferrari Dacrema M et al (2022) Towards feature selection for ranking and classification exploiting quantum annealers. In: Amigó E et al (eds) SIGIR '22: the 45th International ACM SIGIR conference on research and development in information retrieval, Madrid, Spain, July 11 - 15, 2022. ACM, pp 2814–2824. https://doi.org/10.1145/3477495.3531755

Ferrari Dacrema M, Boglio S, Cremonesi P, Jannach D (2021) A troubling analysis of reproducibility and progress in recommender systems research. ACM Trans Inf Syst 39:20:1-20:49. https://doi.org/10.1145/3434185

Glover F, Kochenberger G, Hennig R, Du Y (2022) Quantum bridge analytics I: a tutorial on formulating and using QUBO models. Ann Oper Res 314:141–183. https://doi.org/10.1007/s10479-022-04634-2

Hauke P, Katzgraber HG, Lechner W, Nishimori H, Oliver WD (2020) Perspectives of quantum annealing: methods and implementa-

tions. Rep Prog Phys 83:054401. https://doi.org/10.1088/1361-6633/ab85b8
Hernandez M, Aramon M (2017) Enhancing quantum annealing performance for the molecular similarity problem. Quantum Inf Process 16:133. https://doi.org/10.1007/s11128-017-1586-y
Huang T et al (2023) Benchmarking quantum(-inspired) annealing hardware on practical use cases. IEEE Trans Computers 72:1692–1705. https://doi.org/10.1109/TC.2022.3219257
Ikeda K, Nakamura Y, Humble TS (2019) Application of quantum annealing to nurse scheduling problem. Sci Rep 9. https://doi.org/10.1038/s41598-019-49172-3
Irsigler B, Grass T (2022) The quantum annealing gap and quench dynamics in the exact cover problem. Quantum 6:624. https://doi.org/10.22331/q-2022-01-18-624
Jiang J, Chu C (2023) Classifying and benchmarking quantum annealing algorithms based on quadratic unconstrained binary optimization for solving np-hard problems. IEEE Access 11:104165–104178. https://doi.org/10.1109/ACCESS.2023.3318206
Katzgraber HG, Hamze F, Andrist RS (2014) Glassy chimeras could be blind to quantum speedup: designing better benchmarks for quantum annealing machines. Phys Rev X 4. https://doi.org/10.1103/PhysRevX.4.021008
King AD et al (2024) Computational supremacy in quantum simulation. arXiv e-prints arXiv:2403.00910
Kumar V, Bass G, Tomlin C, III JD (2018) Quantum annealing for combinatorial clustering. Quantum Inf Process 17:39. https://doi.org/10.1007/s11128-017-1809-2
Lucas A (2014) Ising formulations of many np problems. Front Phys 2. https://doi.org/10.3389/fphy.2014.00005
Mandrà S, Zhu Z, Wang W, Perdomo-Ortiz A, Katzgraber HG (2016) Strengths and weaknesses of weak-strong cluster problems: a detailed overview of state-of-the-art classical heuristics versus quantum approaches. Phys Rev A 94. https://doi.org/10.1103/PhysRevA.94.022337
Micheletti C, Hauke P, Faccioli P (2021) Polymer physics by quantum computing. Phys Rev Lett 127. https://doi.org/10.1103/PhysRevLett.127.080501
Mirkarimi P, Hoyle DC, Williams R, Chancellor N (2024) Experimental demonstration of improved quantum optimization with linear Ising penalties. https://doi.org/10.48550/arXiv.2404.05476
Morita S, Nishimori H (2008) Mathematical foundation of quantum annealing. J Math Phys 49:125210. https://doi.org/10.1063/1.2995837
Mott A, Job J, Vlimant JR, Lidar D, Spiropulu M (2017) Solving a Higgs optimization problem with quantum annealing for machine learning. Nature 550:375–379. https://doi.org/10.1038/nature24047
Negre CF, Ushijima-Mwesigwa H, Mniszewski SM (2020) Detecting multiple communities using quantum annealing on the d-wave system. PLoS ONE 15. https://doi.org/10.1371/journal.pone.0227538
Nembrini R, Carugno C, Ferrari Dacrema M, Cremonesi P (2022) Towards recommender systems with community detection and quantum computing. In: Golbeck J et al (eds) RecSys '22: Sixteenth ACM conference on recommender systems, Seattle, WA, USA, September 18 - 23, 2022. ACM, pp 579–585. https://doi.org/10.1145/3523227.3551478
Nembrini R, Ferrari Dacrema M, Cremonesi P (2021) Feature selection for recommender systems with quantum computing. Entropy 23(8):970. https://doi.org/10.3390/e23080970
Neukart F, Dollen DV, Seidel C (2018) Quantum-assisted cluster analysis on a quantum annealing device. Front Phys 6. https://www.frontiersin.org/articles/10.3389/fphy.2018.00055
Neukart F, Von Dollen D, Seidel C, Compostella G (2018) Quantum-enhanced reinforcement learning for finite-episode games with discrete state spaces. Front Phys 5. https://doi.org/10.3389/fphy.2017.00071
Neven H, Denchev VS, Rose G, Macready WG (2009) Training a large scale classifier with the quantum adiabatic algorithm. CoRR abs/0912.0779, http://arxiv.org/abs/0912.0779
Ohzeki M (2020) Breaking limitation of quantum annealer in solving optimization problems under constraints. CoRR abs/2002.05298. https://arxiv.org/abs/2002.05298
O'Malley D, Vesselinov VV, Alexandrov BS, Alexandrov LB (2017) Nonnegative/binary matrix factorization with a d-wave quantum annealer. CoRR abs/1704.01605. http://arxiv.org/abs/1704.01605
Ottaviani D, Amendola A (2018) Low rank non-negative matrix factorization with d-wave 2000q. https://doi.org/10.48550/arXiv.1808.08721
Rieffel EG et al (2015) A case study in programming a quantum annealer for hard operational planning problems. Quantum Inf Process 14:1–36. https://doi.org/10.1007/s11128-014-0892-x
Shannon CE (1948) A mathematical theory of communication. Bell Syst Tech J 27:379–423. https://doi.org/10.1002/j.1538-7305.1948.tb01338.x
Shehzad F, Jannach D (2023) Everyone's a winner! on hyperparameter tuning of recommendation models. In: Zhang J et al (eds) Proceedings of the 17th ACM conference on recommender systems, RecSys 2023, Singapore, September 18-22, 2023. ACM, pp 652–657. https://doi.org/10.1145/3604915.3609488
Snoek J, Larochelle H, Adams RP (2012) Practical Bayesian optimization of machine learning algorithms. https://doi.org/10.48550/arXiv.1206.2944
Stella L, Santoro GE, Tosatti E (2005) Optimization by quantum annealing: lessons from simple cases. Phys Rev B 72:014303. https://doi.org/10.1103/PhysRevB.72.014303
Stollenwerk T, Lobe E, Jung M (2017) Flight gate assignment with a quantum annealer. In: Feld S, Linnhoff-Popien C (eds) Quantum technology and optimization problems - first international workshop, QTOP@NetSys 2019, Munich, Germany, March 18, 2019, Proceedings, vol 11413 of Lecture Notes in Computer Science. Springer, pp 99–110. https://doi.org/10.1007/978-3-030-14082-3_9
Streif M, Neukart F, Leib M (2019) Solving quantum chemistry problems with a d-wave quantum annealer. https://doi.org/10.48550/arXiv.1811.05256
Victoria AH, Maragatham G (2021) Automatic tuning of hyperparameters using Bayesian optimization. Evol Syst 12:217–223. https://doi.org/10.1007/s12530-020-09345-2
Willsch D, Willsch M, Raedt HD, Michielsen K (2020) Support vector machines on the d-wave quantum annealer. Comput Phys Commun 248:107006. https://doi.org/10.1016/j.cpc.2019.107006
Xia R, Bian T, Kais S (2018) Electronic structure calculations and the Ising Hamiltonian. J Phys Chem B 122:3384–3395. https://doi.org/10.1021/acs.jpcb.7b10371
Yarkoni S, Raponi E, Bäck T, Schmitt S (2022) Quantum annealing for industry applications: introduction and review. Rep Prog Phys 85:104001. https://doi.org/10.1088/1361-6633/ac8c54