# Learning QUBO Models for Quantum Annealing: A Constraint-based Approach

Florian Richoux[1,3], Jean-François Baffier[2,3], and Philippe Codognet[3]

[1] AIST, Tokyo, Japan
[2] IIJ Research Lab, Tokyo, Japan
[3] JFLI, CNRS / Sorbonne University / University of Tokyo, Tokyo, Japan
florian@richoux.fr jf@baffier.fr codognet@is.s.u-tokyo.ac.jp

**Abstract.** Quantum Annealing is an optimization process taking advantage of quantum tunneling to search for the global optimum of an optimization problem, although, being a heuristic method, there is no guarantee to find the global optimum. Optimization problems solved by a Quantum Annealer machine are modeled as Quadratic Unconstrained Binary Optimization (QUBO) problems. Combinatorial optimization problems, where variables take discrete values and the optimization is under constraints, can also be modeled as QUBO problems to benefit from Quantum Annealing power. However, defining quadratic penalty functions representing constraints within the QUBO framework can be a complex task. In this paper, we propose a method to learn from data constraint representations as a combination of patterns we isolated in $Q$ matrices modeling optimization problems and their constraint penalty functions. We actually model this learning problem as a combinatorial optimization problem itself. We propose two experimental protocols to illustrate the strengths of our method: its scalability, where correct pattern combinations learned over data from a small constraint instance scale to large instances of the same constraint, and its robustness, where correct pattern combinations can be learned over very scarce data, composed of about 10 training elements only.

**Keywords:** Quantum Annealing · QUBO · Machine Learning · Constrained Optimization Problems · Constraint Satisfaction Problems

## 1 Introduction

As Quantum Computing is getting more real with the effective development of quantum processors, one can distinguish two approaches: the *gate-based paradigm*, in which the main industrial players such as IBM, Google, Intel and many start-up companies (IonQ, Rigetti, IQM, Pasqal, ...) have developed systems with up to a few hundreds of qubits, and the *adiabatic computation paradigm*, in which companies like D-Wave Systems have developed systems with thousands of qubits.

Quantum Annealing (QA) is an instance of adiabatic computation that is interesting in the current Noisy Intermediate-Scale Quantum (NISQ) era, and it

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-36030-5_12

has been applied in the field of combinatorial optimization. Indeed, combinatorial problems can be modeled as Quadratic Unconstrained Binary Optimization (QUBO) as input language and solved by QA systems.

More complex Constrained Optimization and Constraint Satisfaction Problems, coming from the Constraint Programming or Operations Research domains, are now being tackled with QUBO modeling and QA solving [6], although the size of the current QA machines still prevent experiments on large instances. Nevertheless, an interesting issue which is appearing is the modeling of complex problems in the constraint-based approach, and, although it is easy to add penalties into the objective function to represent simple constraints appearing in the problem, it could be difficult to express in a QUBO formulation the penalties corresponding to complex constraints as found in the Constraint Programming paradigm. Indeed, if some constraints are such as the *one-hot* or the *permutation (two-way one-hot)* constraints are easy to represent in QUBO, as we illustrate in Section 3, this is not the case for more general constraints. Many classical combinatorial problems are usually modeled with integer variables and constraints over those integer variables. Thus, to transform those models into QUBO, one has first to encode integer variables by binary variables (this is not difficult) and then to transform the constraints over integer values as penalties over binary variables, which may not be obvious at all. Therefore, an approach that would automatically create the QUBO penalties corresponding to integer constraints would be valuable. We can do that by learning the QUBO matrix representation corresponding to a constraint from the solution and non-solution candidates.

Learning constraints from data has been explored in different directions. Paulus et al [18] proposes to integrate a combinatorial optimization module directly into a neural network as a layer, learning both the constraints and their costs from data. Another approach is proposed by Kumar et al [14], where the constraints and the objective function of Mixed-Integer Linear Programs are learned from data. However, unlike our work, these two papers deal with linear constraints only, and they both learn constraints on a fixed number of variables. In comparison, our method can handle linear and non-linear constraints, and it learns a constraint representation that is independent of the number of variables. This work is inspired from [20], where a method to learn error functions representing constraints is proposed. This paper describes a model, named Interpretable Compositional Network, to learn error functions as an interpretable composition of elementary operations, in such a way that the learned compositions are independent of the number of the target constraint. The main difference with this current paper is that error functions are not necessarily quadratic but must verify a structured property upon error values implying a hierarchy among non-solution candidates, which is not necessary in the present QUBO setting.

## 2 Quantum Annealing and QUBO

Quantum Annealing (QA) has been proposed as a concrete form of adiabatic computation more than two decades ago by Kadowaki et al. [12] and Farhi et

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-36030-5_12

al. [8], and takes advantage of the physical phenomenon of *quantum tunneling*, allowing to traverse energy barriers in the energy landscape as long as they are not too large [23,19]. QA has gained momentum in the last decade with the development of special hardware based on QA, such as the quantum computers of D-Wave Systems [4,16] and, more recently, the so-called "quantum-inspired" systems which are realized with classical (non-quantum) electronics by Fujitsu [1], Hitachi [24], Toshiba [11] or Fixstars Amplify [15]. These systems are sometimes referred to as *Ising Machines*, as they can solve problems stated as Hamiltonians in the Ising model and are aimed to solve a large class of combinatorial problems [22,17], including industrial applications [25].

Interestingly, such a formulation is equivalent to the modeling in Quadratic Unconstrained Binary Optimization, a formalism whose roots go back to pseudo-binary optimization in the late 60's and which has been proposed as a simple but powerful modeling language for combinatorial problems about 15 years ago [2]. QUBO is now seen as a general modeling language for a variety of combinatorial problems [13,10]. For these reasons, QUBO has become the standard input language for Ising machines.

Simply put, a QUBO problem is given by a vector of $n$ binary variables $x_1, \cdots, x_n$ and a quadratic expression over $x_1, \cdots, x_n$ that has to be minimized, which, without loss of generality, is of the form $\sum_{i \leq j} q_{ij} x_i x_j$. Therefore, a QUBO problem is determined by a vector $x$ of $n$ binary decision variables and an upper triangular $n \times n$ square matrix $Q$ with coefficients $q_{ij}$. The QUBO problem can thus be written: minimize $y = x^T Q x$, where $x^T$ is the transpose of $x$.

Moreover, in order to use QUBO to model *Constrained Optimization Problems* (COP) from the field of Operations Research, e.g., the well-known Traveling Salesman problem (TSP) or Quadratic Assignment Problem (QAP), one has to find a way to represent constraint expressions in QUBO models. This can be done by using *penalties* and adding them in the objective function to minimize, that is, as quadratic expressions whose value is minimal when the constraint is satisfied. An easy way to formulate such a penalty is to create a quadratic expression which has value 0 if the constraint is satisfied and a positive value otherwise, representing somehow the degree of violation of the constraint.

Although this works fine for simple constraints, defining the penalties corresponding to complex constraints can, however, become complicated. This is the starting point of our work investigating an automatic manner to generate QUBO penalties corresponding to complex constraints.

## 3 Motivating Example

### 3.1 Basic Example

Consider the problem of coloring the $n$ nodes of a graph with $k$ colors such that no adjacent nodes have the same color. The classical way to model such a problem in QUBO is to consider, for each node $i$ of the graph, $k$ binary variables $x_{ij}$, $j \in \{1, \ldots, k\}$ such that $x_{ij} = 1$ if the node $i$ has the color $j$, and $x_{ij} = 0$ otherwise.

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-36030-5_12

$$Q_c = \begin{pmatrix} -1 & 2 & 2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 2 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 2 & 2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \end{pmatrix} \qquad Q'_c = \begin{pmatrix} -1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & -1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & -1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & -1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \end{pmatrix}$$

Fig. 1: QUBO matrices corresponding to the penalties in Examples 1 and 2

Then, one has to devise an objective function that will be minimal when adjacent nodes have different colors. But the problem variables are also subject to the constraints that each node has only a single color, i.e., that $\sum_1^k x_{ij} = 1$.

This is the well-known *one-hot* constraint: among $k$ Boolean variables, exactly one has to be equal to 1 and others have to be equal to 0. It is classically used for the representation of integer variables with a domain of size $n$ by $n$ Boolean variables, although other encoding such as domain-wall or unary are possible [5,7].

Let us remark that $\sum_{j=1}^k x_{ij} = 1 \iff (\sum_{j=1}^k x_{ij} - 1)^2 = 0$, and by developing this expression, a quadratic penalty expression is obtained:

$$2\sum_{i=1}^{n}\sum_{j<j'} x_{ij}x_{ij'} - \sum_{i=1}^{n}\sum_{j=1}^{k} x_{ij}$$

This penalty has to be added to the QUBO objective function of the original problem in order to enforce the original *one-hot* constraint of the initial problem.

**Example 1** Consider a QUBO model with 9 binary variables $x_{ij}$, $i \in \{1,2,3\}$, $j \in \{1,2,3\}$, and an objective function $f$ to minimize over $x_{ij}$, subject to the three one-hot constraints: $\sum_{j=1}^3 x_{1j} = 1,\ \sum_{j=1}^3 x_{2j} = 1,\ \sum_{j=1}^3 x_{3j} = 1$.

The $9 \times 9$ QUBO matrix $Q$ can be decomposed as the sum $Q = Q_o + Q_c$, with $Q_o$ being the $9 \times 9$ matrix corresponding to the objective function $f$, and $Q_c$ the $9 \times 9$ matrix corresponding to the three one-hot constraints. The $Q_c$ matrix representing the penalty is depicted in Figure 1.

The above transformation is straightforward, and the corresponding QUBO penalty is easy to derive mathematically from the initial constraint (as is the QUBO matrix), but this might not always the case.

More interestingly, we can see that the $9 \times 9$ matrix corresponding to the penalty part of the QUBO model (i.e., the constraint part of the initial problem) shows a particular structure with $3 \times 3$ submatrix around the diagonal. Could the penalty part of the QUBO matrix representing constraints coming from integer problems be seen as a combination of basic patterns such as submatrices? If so,

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-36030-5_12

could we *learn* automatically such matrix representation? We will see that the answer to both questions is "yes".

### 3.2 A More Complex Example

Let us consider now a slightly more complex example: *permutation* constraints. Many classical combinatorial optimization problems, such as the well-known Traveling Salesman Problem (TSP) and the Quadratic Assignment Problem (QAP), or Constraint Satisfaction problems such as the N-queens and Magic Square puzzles are usually modeled with a vector of integer decision variables which are subject to the constraint that each feasible solution forms a permutation. In the Constraint Programming community, such a *permutation constraint* is a special case of the AllDifferent constraint, which has been the subject of large literature and various solving techniques [9].

To enforce that $n$ integer variables $x_i$ with values in $\{1, \ldots, n\}$ represent a permutation, we need to enforce that each value $j \in \{1, \ldots, n\}$ is assigned once and only once. When translated to QUBO, with $n$ binary variables $x_{ij}$, $j \in \{1, \ldots, n\}$, encoding an integer variable $x_i$ of the original problem formulation, this amounts to the so-called *two-way one-hot* constraints: $2 \times n$ one-hot constraints with one set of $n$ constraints corresponding to each of the $n$ variables $x_i$ stating that it can have only one value $k$ and one set of $n$ constraints for each of the $n$ values $k$ stating that it can be assigned to only one variable $x_i$.

These $2 \times n$ one-hot constraints are as follows:

$$\forall i \in \{1, \ldots, n\},\ \sum_{j=1}^{n} x_{ij} = 1 \quad \forall j \in \{1, \ldots, n\},\ \sum_{i=1}^{n} x_{ij} = 1$$

Adding all corresponding penalty expressions together and simplifying the quadratic expression gives the following penalty for the permutation constraint:

$$\sum_{i=1}^{n} \sum_{j<j'} x_{ij} x_{ij'} \ + \ \sum_{j=1}^{n} \sum_{i<i'} x_{ij} x_{i'j} \ - \sum_{i=1}^{n} \sum_{j=1}^{n} x_{ij}$$

**Example 2** Consider a combinatorial problem on 3 integer variables $x_1, x_2, x_3$, with values in $\{1, 2, 3\}$ subject to a permutation constraint and an objective function $f$ to minimize over $x_i$.

If each original integer variable is encoded by 3 binary variables with one-hot encoding, the corresponding QUBO model will have 9 binary variables $x_{ij}$, $i \in \{1, 2, 3\}, j \in \{1, 2, 3\}$, and the $9 \times 9$ QUBO matrix can be decomposed as the sum $Q' = Q'_o + Q'_c$, with $Q'_o$ corresponding to the translation of the objective function $f$ to minimize over $x_{ij}$, and $Q'_c$ corresponding to the permutation constraint. The $Q'_c$ matrix representing the penalty is depicted in Figure 1.

This matrix is somewhat different from the one in Example 1, but a similar pattern of $3 \times 3$ submatrices around the diagonal can be observed, together with patterns on the lines.

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-36030-5_12

## 4 Method Design

The main contribution of this paper is to propose a method to automatically learn from data a pattern composition representing a $Q$ matrix, such that $Q$ corresponds to a target constraint $c$. This method is directly inspired from the method proposed in [20], to learn error functions from data in an interpretable and scalable fashion. We call such a pattern composition a **Q matrix representation**. The data is the training set obtained from an instance of $c$, *i.e.*, the constraint $c$ over a fixed number of variables taking their values over domains of a fixed size.

Let's consider a constraint $c$ and an instance $\iota$ of $c$. A **candidate** of $\iota$ is an assignment of all variables composing $\iota$. A candidate is said to be **positive** if it satisfies $c$, and **negative** otherwise. We denote by $S$ the set of all possible tuples $(x, y)$, where $x$ is a candidate of the constraint instance $\iota$, and $y \in \{0, 1\}$ describes if $x$ is a positive or a negative candidate, such that

$$y = \begin{cases} 1 & \text{if x is a positive candidate} \\ 0 & \text{if x is a negative candidate.} \end{cases}$$

Giving some positive and negative candidates of instance $\iota$ of a constraint $c$, our goal is to learn a pattern composition representing a $Q$ matrix corresponding to $c$, *i.e.*, $Q$ must verify the following property:

$$\forall (x, y) \in S, x^T Q x \text{ is minimal iff } y = 1 \tag{1}$$

Variables of discrete constraints take their value from a domain composed of integers. However, QUBO problems are considering binary variables only. As written in the introduction, there are several ways to convert constraint variables into QUBO variables: unary expansion, binary expansion, etc. In this work, we will only consider one-hot encoding: a constraint variable $x_i$ over a $k$-ary domain will be encoded by a $k$-dimensional binary vector, such that the $j$-th element $x_{ij}$ of this vector is set to true if and only if $x_i = j$ holds.

To represent a discrete constraint, $Q$ can be composed of integers only. If we aim to represent a constraint over $n$ variables $x_i$ taking their value in a $k$-ary domain, then, due to the one-hot encoding of the variables, $Q$ is an upper triangular matrix of size $nk \times nk$. Our method is based on learning a correct combination of submatrix patterns. The one-hot constraint is systematically added into this pattern combination.

$Q$ being an upper triangular matrix, we consider two kinds of submatrices: $k \times k$ triangle submatrices containing the diagonal of $Q$ and representing the properties of a variable $x_i$, and $k \times k$ square submatrices representing properties between two variables $x_i$ and $x_j$, with $i \neq j$.

Our method considers 14 square and 3 triangle submatrix patterns, depicted in Figure 2. Square submatrix patterns can be composed by summing their elements, however, we consider some square submatrix patterns to be mutually exclusive. Indeed, it would make not sense for instance to enforce both the properties $x_i = x_j$ and $x_i \neq x_j$ at the same time. Square patterns 12, 13, 14, and

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-36030-5_12

triangle pattern 3 take some parameters. It is important to keep in mind that submatrices and their patterns are on binary variables.

We can divide square submatrix patterns into 3 categories:

- Comparison patterns from Square 1 to 6, representing some comparison properties between $x_i$ and $x_j$. For instance, $x_i \neq x_j$ (Fig. 2a).
- Position patterns from Square 7 to 11, encoding properties such that the values of the $i$-th and $j$-th variables $x_i$ and $x_j$ depend on their respective position $i$ and $j$. For instance, favoring $x_i = i$ and $x_j = j$ (Fig. 2g).
- Complex patterns for Square 12, 13 and 14. For instance, the repel property described in the next paragraph (Fig. 2l).

Due to the page limitation, we chose not to explain all patterns. Instead, we focus here on the less trivial ones. For Square pattern 14 (Fig. 2n) and triangle pattern 3 (Fig. 2q), $b_{x_i}$ represents the coefficient of the variable $x_i$ in a linear combination $\sum b_{x_i} x_i = a$. Square patterns 12 (Fig. 2l) and 13 (Fig. 2m), respectively called *repel* and *attract*, take a parameter $p$. These patterns look like a diagonal magnetic field with an intense value $p$ in its center (repel) or on its borders (attract), which decays towards the borders (repel) or the center (attract). This is illustrated in Figure 2 with $p = 3$. The intuitive idea behind these patterns is that variables $x_i$ and $x_j$ are repelling or attracting each others, until their values are separated by at least a distance $p$, such that $|x_i - x_j| > p$ holds (repel), or until they are closed enough, below a $k - p$ threshold, such that $|x_i - x_j| < k - p$ holds (attract).

Consider the function $m$ determining if a candidate $x$ is such that $x^T Q x$ is minimal or not:

$$m(x, Q) = \begin{cases} 1 & \text{if } x^T Q x \text{ is minimal,} \\ 0 & \text{otherwise.} \end{cases}$$

Learning a correct combination of submatrix patterns from a training set $X \subseteq S$ is a machine learning problem, but it can also be tackled as a combinatorial problem. We modeled this as a Constrained Optimization Problem (COP) described in Table 1.

We have 14 binary variables to describe which square patterns are combined to give a global pattern for square submatrices, and one variable over a ternary domain indicating which pattern is selected for triangle submatrices. Thus, all square submatrices together will share the same pattern, as well as all triangle submatrices. Although our method still works if we decide to combine different patterns for each submatrix individually, this way of doing has the advantage of learning the $Q$ matrix representation quicker. Moreover, it is motivated by the fact that natural constraints in Constraint Programming usually apply the same property over all variables in their scope (for instance, all variables must be assigned to a different value). If it is not the case, then the target constraint can certainly be decomposed into a series of smaller constraints. For instance, let's assume we work with the constraint $c(x_1, x_2, x_3)$ := "$x_1 + x_2 + x_3 = 5$ such that $x_1 < x_2$ holds". Here, $x_1$ and $x_2$ have a property $x_1 < x_2$ that is not shared by $x_3$. But the constraint $c$ can be decomposed into two constraints

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-36030-5_12

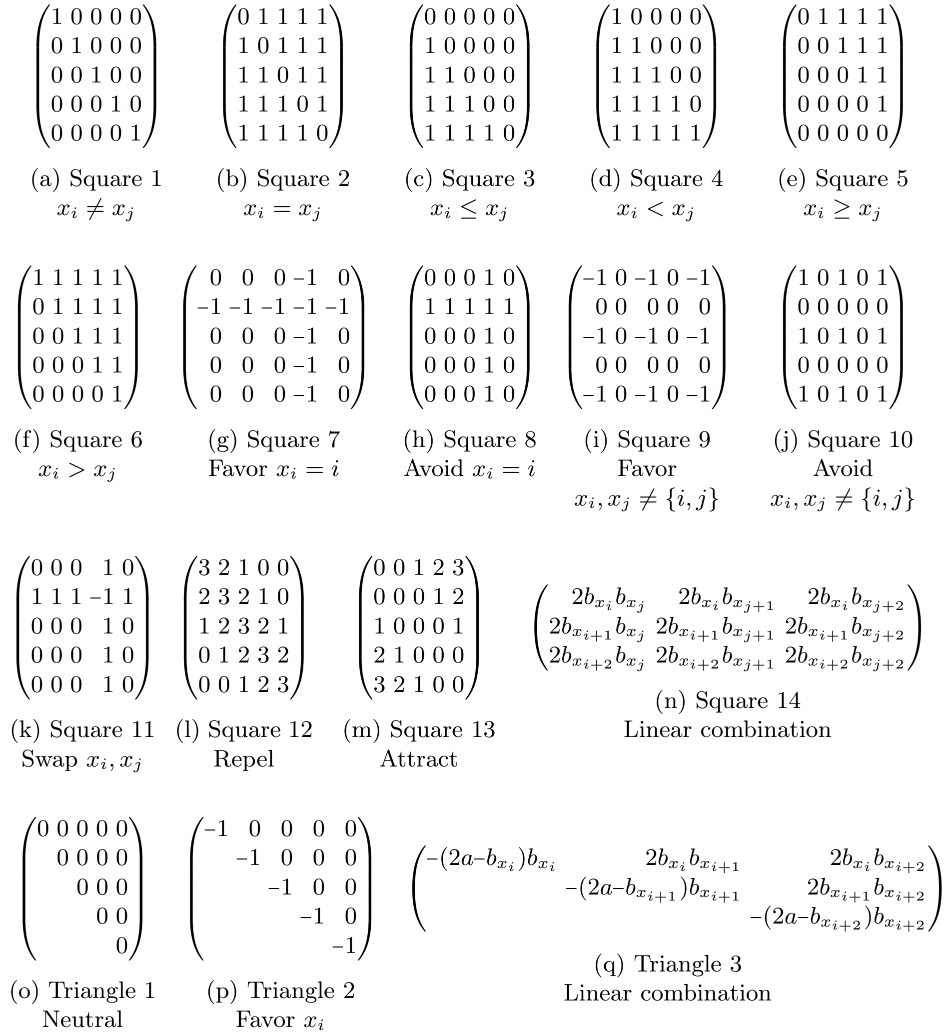

Fig. 2: Submatrix patterns used in our method, with their property. Simple examples are displayed with $k = 5$; complex examples (n) and (q) with $k = 3$.

$c_1(x_1, x_2, x_3) := x_1 + x_2 + x_3 = 5$ and $c_2(x_1, x_2) := x_1 < x_2$, where all variables in their scope share the same properties.

Like expressed above, we forbid some square pattern combinations: We cannot have more than one square pattern from Square 1 to Square 6 in the combination, and couples of patterns Square 7 – Square 8, Square 9 – Square 10, and Square 12 – Square 13 are mutually exclusive. The 4 first constraints in Table 1 are here to forbid such combinations.

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-36030-5_12

Table 1: COP model to learn $Q$ from data $X \subseteq S$

| | |
|---|---|
| Variables | $v_{f_1}, \ldots, v_{f_{14}}, v_h$<br>One variable $v_{f_i}$ for each possible square pattern $i$,<br>a unique variable $v_h$ for the triangle pattern. |
| Domains | $D_{f_i} = \{0, 1\}$, with $1 \leq i \leq 14$<br>$D_h = \{1, 2, 3\}$ |
| Constraints | $\sum_{i=1}^{6} v_{f_i} \leq 1$<br>$v_{f_7} + v_{f_8} \leq 1$<br>$v_{f_9} + v_{f_{10}} \leq 1$<br>$v_{f_{12}} + v_{f_{13}} \leq 1$<br>$\sum_{(x,y) \in X} \lvert m(x, Q) - y \rvert = 0$ |
| Objective function | $\min \sum_{i=1}^{14} v_{f_i}$, minimizing the number of<br>(square) submatrix patterns in the composition. |

The fifth constraint in our COP model makes sure the learned $Q$ matrix representation can correctly handle all positive and negative candidates from the training set $X$.

One strength of this model is its independence regarding the size of the target constraint instance: whatever the size of the data, *i.e.*, the number of variables in candidates or the size of the domains, our model will still be composed of 15 variables to express any pattern combinations describing any constraints. This makes our model scalable, allowing the learning of a $Q$ matrix representation over a small instance of a constraint $c$ that is valid for all instance sizes of the same constraint, as shown in Experiment 1.

## 5 Experiments

To show the versatility of our method, we tested it on five different constraints: AllDifferent, Ordered, LinearSum, NoOverlap1D, and Channel. Following the XCSP$^3$-core specifications[†] [3], those global constraints belong to four major constraint families: Comparison (AllDifferent and Ordered), Counting/Summing (LinearSum), Packing/Scheduling (NoOverlap1D) and Connection (Channel). These constraints are also among the 25 most popular and common constraints [3]. We give a brief description of those five constraints below:

- **AllDifferent** ensures that variables must all be assigned to different values.
- **Ordered** ensures that an assignment of $n$ variables $(x_1, \ldots, x_n)$ must be ordered, given a total order. In this paper, we choose the total order $\leq$. Thus, for all indices $i, j \in \{1, n\}$, $i < j$ implies $x_i \leq x_j$.
- **LinearSum** ensures that the equation $x_1 + x_2 + \ldots + x_n = p$ holds, with the parameter $p$ a given integer.

[†] see also http://xcsp.org/specifications

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-36030-5_12

- **NoOverlap1D** considers variables as tasks, starting from a certain time (their value) and each with a given length $p$ (their parameter). The constraint ensures that no tasks are overlapping, *i.e.*, for all indices $i, j \in \{1, n\}$ with $n$ the number of variables, we have $x_i + p_i \leq x_j$ or $x_j + p_j \leq x_i$. To have a simpler code, we have considered in our system that all tasks have the same length $p$.
- **Channel** ensures that the $i$-th variable $x_i$ assigned to $j$ with $j \neq i$ implies that $x_j$ is assigned to $i$. In other words, Channel accepts all permutations of the vector $(1, \ldots, n)$ such that each variable has been swapped with another variable at most once.

### 5.1 Experimental protocols

We set up two different experimental protocols to show the scalability, the robustness, and more globally, the efficiency of our method.

Like presented in Section 4, learning pattern compositions of $Q$ matrices from data is handled as a combinatorial optimization problem. To model and solve this problem, we use the framework GHOST [21] which runs a stochastic local search solver to solve problems. Due to this stochastic solving, all learning and testing have been done 100 times, but over the same pre-computed training sets, to not let the randomness of sampled sets impact the results in some way. Training and test sets that are too large to be complete have been pre-computed using Latin hypercube sampling to have a good diversity among drawn candidates. These sets have been generated such that they contain an equal number of positive and negative candidates. GHOST's solver requires a timeout, such that it tries to improve the best solution found until it reaches the given timeout. We set it to 1 second. We did not fine-tune the solver parameters, running our experiments with their default values. In addition to our two experiments, we did 100 runs for each training set of Experiment 1 and 2 disabling the objective function, to see how fast our method can find a correct $Q$ matrix representation. Indeed, without an objective function, GHOST's solver does not take into account the timeout and halts as soon as it finds a solution satisfying all constraints in the COP model.

We have hold-out test sets of assignments from larger dimensions to evaluate the quality of our learned $Q$ matrix representations. We did not re-run batches of experiments to keep the ones with the best results, as it should always be the case with such experimental protocols.

All experiments have been done on a computer with a Core i9 9900 CPU and 32 GB of RAM, running on Ubuntu 22.04.2. Programs have been compiled with GCC with the 03 optimization option. Our entire system, its C++ source code, and experimental setups are accessible on Zenodo[§] and GitHub.

**Experiment 1: scalability** One of the key-points of our method is that we can learn the pattern composition of a $Q$ matrix from data about a small instance

[§]https://doi.org/10.5281/zenodo.7800168

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-36030-5_12

of a constraint $c$, *i.e.*, over few variables and small domains, which gives us the blueprint to build $Q$ matrices handling large instances of $c$.

In this experiment, we learn $Q$ matrix representations upon complete training sets composed of all possible tuples $(x, y)$, with $x$ a candidate of the target constraint instance and $y$ the binary value indicating if $x$ is a solution of the constraint or not, as explained at the beginning of Section 4. These small, complete training sets are built considering constraint instances with 4 variables over domains of size 4, giving complete training sets with 256 candidates, except for the constraint NoOverlap1D for which such an instance size does not make sense. The training set for NoOverlap1D takes into account 3 variables over domains of size 7 (and with a parameter $p = 2$ for the length of each task), leading to a training set with $7^3 = 343$ candidates. The parameter $p$ of the LinearSum constraint instance was fixed to 10, giving the constraint $\sum_{i=1}^{4} x_i = 10$, with $x_i \in \{1, \ldots, 4\}, 1 \leq i \leq 4$.

To test both the scalability of our method, we test the learned matrix representations over significantly larger constraint instances, with 30 variables over domains of size 30. Indeed, a learned composition can be represented by a vector of 15 elements, one for each variable of our COP model, independently of the size of the $Q$ matrix and the constraint instance it represents. Those test sets containing too many candidates ($30^{30} \simeq 2 \times 10^{44}$) to be fully tested or even fully generated, we build them in order to get exactly 10,000 build positive and 10,000 drawn negative candidates, giving 20,000 unique candidates. Once again, we need to make an exception for NoOverlap1D due to the nature of this constraint. Its test set is then composed of 10,000 built positive and 10,000 drawn negative candidates of a constraint instance with 20 variables and domains of 160 elements (with $p = 6$), giving a space of $160^{20} \simeq 1.2 \times 10^{44}$ candidates. $Q$ matrices built for these test sets are then of size $900 \times 900$, except for NoOverlap1D which has a $Q$ matrix of size $3200 \times 3200$, going to the limit of what current Quantum Annealing machines can handle nowadays. All test sets are pre-generated, *i.e.*, sampled once and kept for all experiments.

**Experiment 2: robustness** In Experiment 1, we learn pattern compositions of $Q$ matrices to represent constraint instances over complete training sets, *i.e.*, composed of all possible candidates. However, providing such complete sets to learn $Q$ matrix representations may not always be convenient. To test the robustness of our method, we learn $Q$ matrix representations over training sets of constraint instances that are too large to be completely generated in a reasonable time: 12 variables over domains of 12 elements ($12^{12} \simeq 9 \times 10^{12}$). Instead, we sample 5 positive and 5 negative candidates only in such spaces, then we test the learned $Q$ matrix representations on the same test sets than for Experiment 1. For NoOverlap1D, we sample 5 positive and 5 negative candidates of an instance with 8 variables over domains of 35 elements (and $p = 3$), leading to a space composed of $35^8 \simeq 2.2 \times 10^{12}$ candidates.

In addition, we also tested how restricted training sets can be until we observe significant efficiency drops. For this, we repeated this experimental protocol with

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-36030-5_12

Table 2: Success rates over 100 runs on test sets. Timeout is fixed at 1s for each run.

| Experiment | AllDifferent | Ordered | LinearSum | NoOverlap1D | Channel |
|---|---|---|---|---|---|
| 1 | 100 | 100 | 100 | 100 | 100 |
| 2 | 100 | 100 | 100 | 97 | 100 |

training sets composed of 2, 4, 6, 8, 10 and 12 candidates, each time with half positive and half negative candidates. Like test sets, these training sets are pre-generated.

### 5.2 Experimental results

Table 2 shows the number of times, over 100 runs and for each target constraint, a $Q$ matrix representation has been learned and corresponds to a correct $Q$ matrix, *i.e.*, satisfying the property of Equation 1 over tuples $(x, y)$ of the constraint test set. This first line corresponds to the results of Experiment 1, demonstrating that our method perfectly scales, learning $Q$ matrix representation over small training sets of about 300 candidates and successfully tested over 20,000 randomly drawn candidates from a huge candidate space of about $2 \times 10^{44}$ candidates. The second line is the results of Experiment 2. It shows that our method is able to learn correct $Q$ matrix representations from very sparse data: here, we learn $Q$ matrix representation from 5 positive and 5 negative candidates, randomly drawn from candidate spaces of about $9 \times 10^{12}$ candidates. All learned $Q$ matrices shown themselves to be correct on test sets, except 3 times for NoOverlap1D. Indeed, for these 3 times, the solver has been trapped into some local optimum, finding a correct pattern composition for the training set but containing unnecessary patterns, that lead to incorrectly handled candidates in the NoOverlap1D test set. More specifically, almost all positive candidates from the NoOverlap1D test set were incorrectly handled and considered as negative candidates. Having false negatives is indeed more frequent than false positive because a positive candidate $x_p$ with any values of $x_p^T Q x_p$ above the expected minimal value will be considered to be a negative candidate, but mistaking a negative candidate $x_n$ as a positive one requires having $x_n^T Q x_n$ to be equals to the expected minimal value. Having a value $x_n^T Q x_n$ below the expected minimal value in the test set is unlikely because it would imply that all positive candidates would be considered as false negatives.

These three incorrectly learned $Q$ matrix representations for NoOverlap1D is not a serious problem in practice: users can get around this problem either by considering more candidates in the training set, or setting a timeout longer than 1 second to let the solver more time to find an optimal solution, or finally in the worst case, running again the learning. Since learning a $Q$ matrix representation only takes one second in our current setup, users can easily afford re-learning a $Q$ matrix representation if the first learning outputs an incorrect one.

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-36030-5_12

Table 3: Success rate over 100 runs on test sets, regarding the size of the incomplete training sets. Timeout is fixed at 1s for each run.

| Nb candidates | AllDifferent | Ordered | LinearSum | NoOverlap1D | Channel |
|---|---|---|---|---|---|
| 12 | 100 | 100 | 100 | 92 | 100 |
| 10 | 100 | 100 | 100 | 96 | 100 |
| 8 | 37 | 100 | 100 | 97 | 100 |
| 6 | 33 | 100 | 100 | 92 | 10 |
| 4 | 27 | 100 | 100 | 39 | 10 |
| 2 | 0 | 0 | 19 | 8 | 0 |

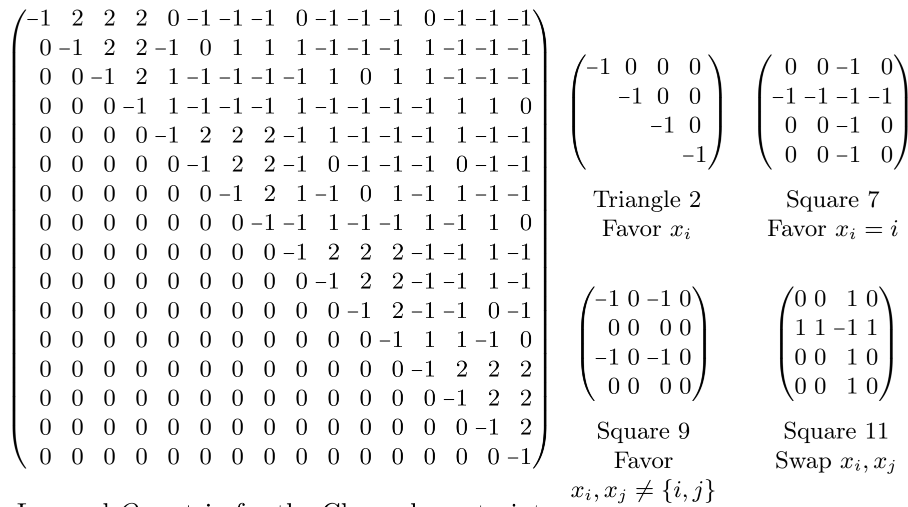


Fig. 3: $Q$ matrix for Channel and patterns in the learned composition.

In Experiment 2, we learned $Q$ matrix representation from training sets composed of 5 positive and 5 negative candidates. We tested how many candidates were needed to correctly learn pattern compositions for our different constraints. Table 3 sums up these trials, showing the success rate over 100 runs of representation learning with training sets of size $n$, with $\frac{n}{2}$ positive and $\frac{n}{2}$ negative candidates. Unsurprisingly, we can see no correct representations can be learned with one positive and one negative candidate only. However, four balanced candidates is sufficient to perfectly learn the $Q$ matrix representation of the constraints Ordered and LinearSum.

Figure 3 illustrates a $Q$ matrix obtained with our method, for the Channel constraint with 4 variables over domains of size 4, and the patterns that are taking part of the learned combination. The one-hot constraint is not depicted as a pattern in this figure, but it is always implicitly added in the combination.

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-36030-5_12

## 6 Conclusion

In this paper, we presented a method to learn a pattern composition representing the penalty part of a QUBO matrix, which can be difficult to do manually for complex constraints. We showed that this can be done with a limited set of examples and, via two experimental protocols, that our method has excellent scalability and robustness properties.

The current limitations of our method is the incapacity to handle ancillary variables. Indeed, some constraints require additional binary variables that do not directly represent integer variables from the target constraint, but are necessary to model some interactions between integer variables. Adding automatically the right number of ancillary variables to handle constraints like Element or NValues would be a natural extension of our work.

A second limitation is the lack of input size-dependent patterns, such as the size of domains, for instance. Such patterns would be greatly helpful to model some constraints like Minimum, seeing their satisfaction depending on some fixed values, without having a combinatorial explosion of the number of patterns that could severely hurt the learning efficiency. However, defining such generic and input size-dependent patterns might be challenging.

## References

1. Aramon, M., Rosenberg, G., Valiante, E., Miyazawa, T., Tamura, H., Katzgraber, H.G.: Physics-inspired optimization for quadratic unconstrained problems using a digital annealer. Frontiers in Physics **7**, 48 (2019)
2. Boros, E., Hammer, P.L., Tavares, G.: Local search heuristics for quadratic unconstrained binary optimization (QUBO). J. Heuristics **13**(2), 99–132 (2007)
3. Boussemart, F., Lecoutre, C., Audemard, G., Piette, C.: XCSP3-core: A format for representing constraint satisfaction/optimization problems. arXiv **abs/2009.00514** (2020)
4. Bunyk, P.I., Hoskinson, E.M., Johnson, M.W., Tolkacheva, E., Altomare, F., Berkley, A.J., Harris, R., Hilton, J.P., Lanting, T., Przybysz, A.J., Whittaker, J.: Architectural considerations in the design of a superconducting quantum annealing processor. IEEE Transactions on Applied Superconductivity **24**(4), 1–10 (2014)
5. Chancellor, N.: Domain wall encoding of discrete variables for quantum annealing and QAOA. Quantum Science and Technology **4**, 045004 (2019)
6. Codognet, P.: Constraint solving by quantum annealing. In: Silla, F., Marques, O. (eds.) ICPP Workshops 2021: 50th International Conference on Parallel Processing, USA, August 9-12, 2021. pp. 25:1–25:10. ACM (2021)
7. Codognet, P.: Domain-wall / unary encoding in QUBO for permutation problems. In: 2022 IEEE International Conference on Quantum Computing and Engineering (QCE). pp. 167–173 (2022)
8. Farhi, E., Goldstone, J., Gutmann, S., Lapan, J., Lundgren, A., Preda, D.: A quantum adiabatic evolution algorithm applied to random instances of an np-complete problem. Science **292**(5516), 472–475 (2001)

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-36030-5_12

9. Gent, I.P., Miguel, I., Nightingale, P.: Generalised arc consistency for the alldifferent constraint: An empirical survey. Artificial Intelligence **172**(18), 1973–2000 (2008)
10. Glover, F.W., Kochenberger, G.A., Du, Y.: Quantum bridge analytics I: a tutorial on formulating and using QUBO models. 4OR **17**(4), 335–371 (2019)
11. Goto, H., Tatsumura, K., Dixon, A.R.: Combinatorial optimization by simulating adiabatic bifurcations in nonlinear hamiltonian systems. Science Advances **5**(4) (2019)
12. Kadowaki, T., Nishimori, H.: Quantum annealing in the transverse Ising model. Phys. Rev. E **58**, 5355–5363 (Nov 1998)
13. Kochenberger, G.A., Hao, J., Glover, F.W., Lewis, M.W., Lu, Z., Wang, H., Wang, Y.: The unconstrained binary quadratic programming problem: a survey. J. Comb. Optim. **28**(1), 58–81 (2014)
14. Kumar, M., Kolb, S., De Raedt, L., Teso, S.: Learning mixed-integer linear programs from contextual examples. arXiv e-prints **abs/2107.07136**, 1–11 (2021)
15. Matsuda, Y.: Research and development of common software platform for ising machines. In: 2020 IEICE General Conference (2020)
16. McGeoch, C.C., Harris, R., Reinhardt, S.P., Bunyk, P.I.: Practical annealing-based quantum computing. Computer **52**(6), 38–46 (2019)
17. Mohseni, N., McMahon, P.L., Byrnes, T.: Ising machines as hardware solvers of combinatorial optimization problems. Nature Rev. Phys. **4**(6), 363–379 (2022)
18. Paulus, A., Rolínek, M., Musil, V., Amos, B., Martius, G.: Comboptnet: Fit the right np-hard problem by learning integer programming constraints. In: Proceedings of the 38th International Conference on Machine Learning (ICML 2021). pp. 8443–8453. PMLR, Online (2021)
19. Rajak, A., Suzuki, S., Dutta, A., Chakrabarti, B.K.: Quantum annealing: an overview. Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences **381**(2241) (dec 2022)
20. Richoux, F., Baffier, J.F.: Automatic error function learning with interpretable compositional networks. Springer Annals of Mathematics and Artificial Intelligence pp. 1–35 (2023)
21. Richoux, F., Uriarte, A., Baffier, J.F.: GHOST: A combinatorial optimization framework for real-time problems. IEEE Transactions on Computational Intelligence and AI in Games **8**(4), 377–388 (2016)
22. Tanahashi, K., Takayanagi, S., Motohashi, T., Tanaka, S.: Application of ising machines and a software development for ising machines. Journal of the Physical Society of Japan **88**(6), 061010 (2019)
23. Tanaka, S., Tamura, R., Chakrabarti, B.K.: Quantum Spin Glasses, Annealing and Computation. Cambridge University Press, USA, 1st edn. (2017)
24. Yamaoka, M., Okuyama, T., Hayashi, M., Yoshimura, C., Takemoto, T.: CMOS annealing machine: an in-memory computing accelerator to process combinatorial optimization problems. In: IEEE Custom Integrated Circuits Conference, Austin, TX, USA, 2019. pp. 1–8. IEEE (2019)
25. Yarkoni, S., Raponi, E., Bäck, T., Schmitt, S.: Quantum annealing for industry applications: introduction and review. Reports on Progress in Physics **85**(10), 104001 (sep 2022)


ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-36030-5_12