From Table Constraints to QUBO Models for Quantum Annealing

Philippe Codognet[1][0000−0002−6254−6389] and Eric Monfroy[2][0000−0001−7970−1368]

[1] JFLI - CNRS / Sorbonne Université / NII, Tokyo, Japan
[2] University of Angers, LERIA, France
codognet@is.s.u-tokyo.ac.jp Eric.Monfroy@univ-angers.fr

Abstract. Quantum optimization has grown rapidly in the past decade, supported by advances in both hardware and software frameworks such as Quadratic Unconstrained Binary Optimization (QUBO). In contrast, Constraint Programming (CP) provides a highly expressive declarative approach for modeling combinatorial problems—an expressiveness that QUBO lacks. Bridging these two worlds requires systematic methods to translate CP models into quantum-compatible formulations. While some automated conversions exist for linear arithmetic and certain global constraints, extensional constraints—capable of expressing any finite-domain relation—have not yet been addressed. This paper introduces a method for automatically translating table constraints, a widely used form of extensional constraint, into QUBO. The conversion is efficient: binary tables require no ancillary variables, and tables with l n-ary tuples require only l additional Boolean variables per element constraint.

Keywords: Constraint Programming · Table Constraint · QUBO Model · Quantum Annealing

1 Introduction

The new domain of *quantum optimization*, that is, the use of quantum computing for solving combinatorial optimization problems, has experienced a significant growth in the last decade, thanks to the developments both in hardware systems, such as the D-Wave quantum computers based on Quantum Annealing [19], and in software frameworks for problem modeling, such as *Quadratic Unconstrained Binary Optimization (QUBO)* [24]. Due to its equivalence with the Ising model and the seminal work of Lucas [23] in modeling combinatorial problems in the Ising model, QUBO has become a standard input language in the Quantum Annealing paradigm but also for the Quantum Approximate Optimization Algorithm (QAOA) in the gate-based paradigm [14]. However QUBO is a relatively low-level framework and complex optimization problems are more easily expressed in higher level approach such as Constraint Programming (CP) [28], which is a declarative approach for addressing combinatorial problems, relying on methods from artificial intelligence, computer science, and operations research. Instead of describing an explicit algorithm for computing the solution to a given

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29918-5_1

problem, CP enables users to express a problem by using decision variables and stating constraints that define the permissible relationships between them.

Reconciling the high-level modeling expressiveness of Constraint Programming with the rapidly evolving capabilities of quantum computing calls for systematic mechanisms to translate CP formulations into quantum-compatible models. In particular, the availability of automatic transformations from Constraint Satisfaction or Constrained Optimization Problems (CSPs/COPs) into Quadratic Unconstrained Binary Optimization (QUBO) models would greatly facilitate the use of quantum and quantum-inspired solvers on problems modeled in a constraint-based approach.

Extensional constraints, such as table constraints [28], are able to represent any finite relation over a set of variables by explicitly enumerating the allowed (or forbidden) tuples. Because the vast majority of Constraint Programming models rely on finite-domain integer variables, the ability to automatically translate table constraints into QUBO formulations is therefore essential. Beyond expressiveness, table constraints also play a key role as a canonical intermediate representation in many modeling and compilation frameworks, making their direct conversion particularly valuable for automated CP-to-QUBO pipelines. A first attempt at translating table constraints into a QUBO formulation was presented in [12]. However, this translation was specifically tailored to a single case – the cyclic bandwidth problem – and applied only to tables of pairs of labels. In this paper we generalize, formalize, and extend the work of [12]. We propose, to the best of our knowledge, the first systematic conversion procedure from table constraints to QUBO. We describe in detail how both binary and general n-ary table constraints can be translated into explicit QUBO penalty matrices. For binary table constraints, the proposed encoding requires no auxiliary variables, leading to compact formulations. For n-ary constraints, we introduce auxiliary variables to preserve correctness and analyze their number, providing precise bounds on the additional modeling overhead induced by the conversion.

This paper is organized as follows. Section 2 presents QUBO and Quantum Annealing, while Section 3 introduces the notions of table constraints. The transformation of table constraints in QUBO is detailed in Section 4, for both binary and n-ary constraints. Several examples illustrating this approach are then described: first, in Section 5, two simple examples, the classical N-queens problem and the traffic lights problem and then a hard combinatorial problem in Section 6, the Costas Array Problem.

2 QUBO and Quantum Annealing

The current development of QUBO is linked to its use as the standard formalism for the modeling of combinatorial problems in the Quantum Annealing paradigm [30]. Although the foundations of the QUBO formalism lie in research into pseudo-Boolean optimization which started in the late 1960's [17, 7], QUBO became established as a general modeling language for combinatorial problems about one decade ago [20], see [15] for a detailed tutorial presentation or [24]

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29918-5_1

for a comprehensive treatment. Today, QUBO serves as the universal interface for quantum and "quantum-inspired" annealing hardware and is a cornerstone in the application of quantum computing to combinatorial optimization.

Consider n Boolean variables $x_1, ..., x_n$, a Quadratic Unconstrained Binary Optimization (QUBO) problem consists in minimizing an *objective function* defined by a quadratic expression over $x_1, ..., x_n$:

$$\sum_{i \leq j} q_{ij} x_i x_j$$

A QUBO problem can therefore be represented by a vector x of n Boolean decision variables and a square $n \times n$ matrix Q with coefficients q_{ij}, and the minimization problem can be written in a matrix format:

$$\text{minimize } y = x^\intercal Q x, \text{where } x^\intercal \text{ is the transpose of } x.$$

When modeling a combinatorial problem in the QUBO paradigm, *constraints* between problem variables (for instance $x \leq y$ or $x + y = 1$) are transformed into *penalties* that are added to the objective function. A penalty is a quadratic expression whose value is minimal when the constraint is satisfied, for instance a penalty can be a quadratic expression which has value zero if the constraint is satisfied and a positive value otherwise. In such a case, the penalty intuitively represents the degree of violation of the constraint, hence the name. However in the general case, as QUBO accepts objective functions to minimize which are not bounded by zero, penalties may have negative values as the minimal value. The key point is that the minimum is achieved only when the original constraint is satisfied.

Let us consider simple examples of a penalty corresponding to a constraint. Consider again the two pseudo-Boolean constraints $x \leq y$ and $x + y = 1$ between two Boolean variables x and y. The penalty corresponding to the constraint $x \leq y$ can be defined as the quadratic expression $x - xy$, while the penalty corresponding to the constraint $x + y = 1$ can be defined as the quadratic expression $1 - x - y + 2xy$, cf. [15]. These are simple examples of quadratic penalties corresponding to constraints between Boolean variables, but the problem of defining quadratic penalties becomes more complex when considering more complex constraints relating integer decision variables.

In general, when added to the objective function of the QUBO problem, the penalties must be weighted with a penalty coefficient p large enough to make this constraint "hard" while the rest of the objective function will be "soft". The problem of defining the correct penalty weights can however become quite complicated [31], as for implementation reasons it is better to have weights which are not too large.

2.1 Encoding Integers into Booleans

There is another point to take care when modeling combinatorial problems in QUBO. Most classical combinatorial problems such as constraint satisfaction

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29918-5_1

or constrained optimization problems are modeled with integers (representing a discrete, finite domain) rather than with Booleans.

Among the most prominent integer-to-Boolean encodings used in the QUBO community are the widely used one-hot encoding [16] and the newer domain-wall/unary representation [8, 9]. For simplicity, we will adopt the one-hot encoding in this paper, but domain-wall/unary encoding could be used as well. Under the one-hot scheme, an integer variable x with n distinct possible values is mapped to n Boolean variables x_i, $i \in \{1, \ldots, n\}$ in such a way that exactly one variable x_i is set to 1, indicating the value assigned to x (i.e., $x_i = 1$ iff $x = i$), and all other variables are set to 0. To guarantee that the variable is assigned a single value, the one-hot representation thus requires to enforce the pseudo-Boolean constraint $\sum_{i=1}^{n} x_i = 1$. In the QUBO framework, this condition can be imposed by adding a quadratic penalty term of the form $-\sum_{i=1}^{n} x_i + \sum_{i \neq j} x_i x_j$ which is added to the original objective function. This penalty evaluates to zero when exactly one variable x_i equals 1 and is strictly positive otherwise, thereby enforcing the one-hot condition through optimization.

In addition to the representation of variables, a number of studies have investigated how standard constraints can be translated into QUBO form, with particular focus on linear constraints expressed as $\sum_{i=1}^{m} a_i x_i = b$ (see, for example, [15]). More recently, research efforts have begun to extend this approach to global constraints, including constraints such as permutation, which have been specifically addressed in [11], for both one-hot and unary/domain-wall schemes, or a learning-based approach to derive the penalty matrix associated with certain types of global constraints [25].

3 Constraint programming and Table constraints

Constraint Programming systems provide general-purpose solvers capable of handling such models. These solvers typically combine exploration techniques such as depth-first search with inference mechanisms, notably constraint propagation, to reduce the search space [3, 28]. Depending on the problem, solvers may also integrate specialized components, including tailored branching strategies and heuristics.

Constraints may involve different classes of variables, such as Boolean variables or bounded integers, and can take a wide variety of forms, ranging from linear or nonlinear arithmetic constraints to symbolic relations. Moreover, constraints can be specified either extensionally, by explicitly enumerating all allowed tuples (e.g., in the form of a truth table), or intensionally, by defining a property or a function that characterizes the valid tuples. Problems are commonly formulated as Constraint Satisfaction Problems (CSPs) or Constrained Optimization Problems (COPs). A CSP consists of a set of variables, their associated domains, and a collection of constraints that restrict allowable assignments. A COP extends this framework by introducing an objective function whose value is to be minimized or maximized. A solution to a CSP satisfies all constraints, while a solution to a COP additionally optimizes the specified objective. Overall,

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29918-5_1

CP offers a flexible and expressive framework that supports both the modeling and the effective resolution of complex combinatorial problems.

3.1 Extensional Constraints

The use of extensional constraints (e.g., tables, decision diagrams, ...) in Constraint Programming dates back more than two decades. Early work already recognized the interest of describing constraints by explicitly listing their allowed combinations of values, rather than relying solely on intensional or arithmetic formulations [6]. In particular, the authors of [4] demonstrated that a n-ary constraint defined by a truth table could be systematically and automatically exploited to derive a constraint solver capable of enforcing generalized arc consistency without losing the structure of the constraint, thereby showing that extensional representations were not merely declarative but could also support effective propagation mechanisms. Then, table constraints became a prominent object of study within the CP community. Subsequent research focused on the design of increasingly efficient propagation algorithms (together with enriched tables to provide a more declarative specification by expressing structural relationships between columns, rather than just listing tuples), leading to a series of advances in generalized arc consistency for table constraints. Among these algorithms, let us cite [22] or STR3, the last variant of the STR algorithms [21].

In parallel, more theoretical investigations clarified the expressive power of extensional constraints and formally established their respective abilities to represent arbitrary finite relations: which representation could be the best-fitted one, and what operations/queries are tractable on these representations [33]. Some more encodings where also proposed (e.g., [32]). Over time, table constraints evolved from a convenient modeling device into a central abstraction in both theory and practice. They are now widely used not only to capture complex or data-driven relations, but also as a uniform representation. This dual role—combining maximal expressiveness with strong algorithmic support—has made extensional constraints a powerful aspect of Constraint Programming.

3.2 Table constraints

Consider $I_1, \ldots, I_n$, n finite sets of integers. A n-tuple over $I_1, \ldots, I_n$ is an element of $I_1 \times \ldots \times I_n$. A table $T \subseteq I_1 \times \ldots \times I_n$ is a finite set of such tuples:

$$T = \{(t_{1,1}, t_{1,2}, \ldots, t_{1,n}), \ldots, (t_{m,1}, t_{m,2}, \ldots, t_{m,n})\}$$

with each $t_{i,j} \in I_j$.

Now consider a n-tuple of finite domain integer variables (i.e., decision variables having a finite number of candidate values) $(x_1, \ldots, x_n)$, each x_i with a domain I'_i. Then, the table constraints we consider are the following:

$$(x_1, \ldots, x_n) \in T$$

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29918-5_1

meaning that the instantiation of the variables x_i's is a n-tuple of T. Conversely, we also consider constraints of the form:

$$(x_1, \ldots, x_n) \notin T$$

meaning that the instantiation of the x_i's is not a n-tuple of T (or is a tuple of $\bar{T} = I'_1 \times \ldots \times I'n \setminus T$). In the following, we consider the same domain $D_{x_1} \times \ldots \times D_{x_n}$ for tuples of variables and $I_1 \times \ldots \times I_n$ for tuples of the table, i.e., $D_{x_1} \times \ldots \times D_{x_n} = I_1 \times \ldots \times I_n$. This largely simplifies the presentation and does not restrict the scope. Indeed, in case they are different, we can consider their intersection $D_{x_1} \times \ldots \times D_{x_n} \cap I_1 \times \ldots \times I_n$ without changing the solutions.

4 Tables constraints in QUBO

4.1 Variable encoding

In the following, we use the one-hot encoding of integers into Booleans for clarity. Each integer variable x in a finite domain $D_x = \{v_1, \ldots, v_n\}$ is represented by n Boolean variables x_i with this meaning: $x_i = 1 \Leftrightarrow x = v_i$, otherwise $x_i = 0$.

Since a variable x has one and only one value, we must also enforce that only one x_i is 1 and that the others are 0. This is achieved by the one-hot constraint $\sum_{i=1}^{n} = 1$ which gives the penalty matrix:

$$P^x_{oh} = \begin{array}{c} \\ x_1 \\ x_2 \\ \vdots \\ x_{n-1} \\ x_n \end{array} \begin{array}{c} \begin{array}{ccccc} x_1 & x_2 & x_3 & \ldots & x_n \end{array} \\ \begin{pmatrix} -1 & 1 & \ldots & \ldots & 1 \\ 1 & -1 & 1 & \ldots & 1 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 1 & \ldots & 1 & -1 & 1 \\ 1 & \ldots & \ldots & 1 & -1 \end{pmatrix} \end{array}$$

4.2 Tables for binary constraints

In this section, we fix $n = 2$, and thus, tables of couples. Consider a couple of variables (x, y) such that the domain of x is $D_x = \{v_1, \ldots, v_k\}$ and the one of y is $D_y = \{w_1, \ldots, w_l\}$. Consider a table $T = \{(v_i, w_j) \mid v_i \in D_x \wedge w_j \in D_y\}$.

Element of a table. Consider the CSP constraint $(x, y) \in T$. Assume a one-hot encoding of x by $x_1, \ldots, x_k$ and y by $y_1, \ldots, y_l$ as defined above. The pseudo-Boolean constraint associated to the constraint $(x, y) \in T$ is thus:

$$\sum_{(v_i, w_j) \in T} x_i y_j \geq 1$$

and the corresponding penalty function (to minimize) is thus:

$$\sum_{(v_i, w_j) \in T} -x_i y_j$$

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29918-5_1

Intuitively, we reward the couples of values that are in the table by a "penalty" of -1 (recall that we are in a minimization problem).

Then, the penalty sub-matrix for this table constraint is:

$$Q_T = \begin{array}{c} \\ x_1 \\ x_2 \\ \vdots \\ x_k \end{array} \begin{array}{c} \begin{array}{cccc} y_1 & y_2 & \cdots & y_l \end{array} \\ \begin{pmatrix} q(1,1) & q(1,2) & \dots & q(1,l) \\ q(2,1) & q(2,2) & \dots & q(2,l) \\ \vdots & \vdots & \dots & \vdots \\ q(k,1) & q(k,2) & \dots & q(k,l) \end{pmatrix} \end{array} \text{ with } q(i,j) = \begin{cases} -1 & \text{if } (v_i, w_j) \in T \\ 0 & \text{otherwise} \end{cases}$$

Note that this matrix may not be square if the size of D_x is not the same as the size of D_y. Finally, the penalty matrix, taking into account the one-hot constraints for x and y, and the constraint $(x, y) \in T$, is made of 4 sub-matrices that are defined above (where ${Q_T}^\intercal$ denotes the transpose of Q_T):

$$\begin{array}{c} \\ x_1 \\ \vdots \\ x_k \\ y_1 \\ \vdots \\ y_l \end{array} \begin{array}{c} \begin{array}{cc} x_1 \dots x_k & y_1 \dots y_l \end{array} \\ \begin{pmatrix} P^x_{oh} & Q_T \\ {Q_T}^\intercal & P^y_{oh} \end{pmatrix} \end{array}$$

Not element of a table. Consider now the CSP constraint $(x, y) \notin T$. We consider the equivalence $(x, y) \notin T \iff (x, y) \in \bar{T}$, where $\bar{T}$ is the complement of T. Then we apply the previous process.

4.3 Tables for n-ary constraints

Consider $n > 2$ finite integer domains $D_i = \{d_{i,1}, \cdots d_{i,l_i}\}$. A table T is now a subset of $D_1 \times \ldots \times D_n$ and is thus a set of l n-tuples of the form $(d_{1,i_1}, \cdots, d_{n,i_n})$ with $i_j \in [1..l_j]$ for all j.

Consider n finite domain variables $x_1, \ldots, x_n$ each x_i associated to the domain D_i. Assume now a constraint $(x_1, \ldots, x_n) \in T$, meaning that the instantiation of the variables $(x_1, \ldots, x_n)$ must be a tuple of T.

To convert this constraint into QUBO, first we represent each variable x_i by a set of Boolean variables $x_{i,j}$ with the same semantics as before: the Boolean variable $x_{i,j} = 1$ iff the finite domain variable $x_i = d_{i,j}$, and the $x_{i,j}$ are equipped with a one-hot constraint $\sum_{j=1}^{l_i} x_{i,j} = 1$ (i.e., x_i has one and only one value).

Extending from the scheme defined for binary constraints, the intuitive conversion of the table constraint $(x_1, \ldots, x_n) \in T$ is thus:

$$\sum_{(i_1,\ldots,i_n)\in T} x_{1,i_1} \cdots x_{n,i_n} \geq 1$$

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29918-5_1

However, we need now some extra-work to derive the QUBO penalty term, as each monomial term is here of order n and not quadratic. Although it is well-known that all polynomial expressions over Booleans can be reduced to equivalent quadratic expressions, transforming a PUBO/HOBO model (Polynomial Unconstrained Boolean Optimization / Higher Order Boolean Optimization) into a QUBO model is not direct [2]. Whether applying specific techniques to quadratize cubic [27], or quadratic [18] monomials or using the general scheme developed in [26], this *quadratization* process always amounts to the introduction of ancillary variables, which is problematic in the quantum setting where each QUBO variable will be represented by a (logical) qubits and qubits are a rather scarce ressource. In the case of the conversion of a table constraint, each monomial $x_{1,i_1}.\cdots.x_{n,i_n}$ would need $n-1$ extra variables e_j such that: $e_1 = x_{1,i_1}x_{2,i_2}$, $e_2 = e_1x_{3,i_3}$, etc. when using the simple Rosenberg method [27] or at least $\lceil log(n) \rceil - 1$ extra variable when using the more complex method proposed in [26]. Observe that one has to introduce such a number of ancillary variables for each entry in the table.

Instead of complex transformations of higher-order monomials, we propose a simple and direct method, inspired by the transformation of n-ary constraint networks into binary constraint networks [29]. To this end, we consider an integer variable t with domain $1..l$, with $l = |T|$, and we associate a unique number in $1..l$ to each tuple of T. Thus, each tuple $(d_{1,i_1}, \cdots, d_{n,i_n}) \in T$ is extended into a tuple $(d_{1,i_1}, \cdots, d_{n,i_n}, j)$, and T' is the set of such tuples built from T. The variable t can also be converted into k Boolean variables t_i either by one-hot encoding and therefore $k = l$, or by binary (log) encoding, using thus $k = \lceil log(l) \rceil$.

We now consider n binary tables T_i defined by:

$$(d_1, \cdots, d_n, j) \in T' \Leftrightarrow (d_1, j) \in T_1 \ \wedge \ \cdots \ \wedge \ (d_n, j) \in T_n$$

The penalty we consider for this conversion is as follows:

$$\sum_{k=1}^{n} \sum_{(d_{k,i},j) \in T_k} -x_{k,i}t_j$$

Minimizing this penalty will ensure that the original table constraint is satisfied.

In case of a not element constraint, i.e., $(x_1, \ldots, x_n) \notin T$, the technique is exactly the same as what was done for the binary case. We consider the equivalence $(x_1, \ldots, x_n) \notin T \iff (x_1, \ldots, x_n) \in \bar{T}$, where $\bar{T}$ is the complement of T. Then we apply the above process. Observe nevertheless that T and $\bar{T}$ might not be of the same size, and thus the number of extra-variables may be significantly different.

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29918-5_1

5 Illustrative examples

5.1 N-Queens

The well-known n-queen problem (Problem 054 from the CSPlib [1]) consists in placing n queens on a $n \times n$ chessboard such that no two queens attack each other. This problem can be modeled directly in QUBO as an optimization problem with Boolean decision variables and inequation constraints, but we will show here, as an illustration of our method, how to derive a QUBO model from a constraint satisfaction formulation. The classical formulation relies on a constraint-based model with finite-domain variables which uses n variables $q_1, \ldots, q_n$, each taking values from the domain $\{1, \ldots, n\}$, where $q_i = j$ denotes that the queen placed in column i occupies row j.

The problem is then defined by a set of table constraints enforcing that no two queens are on the same row, and that not 2 queens at distance d share the same diagonals:

- $\forall 1 \leq i < j \leq n,\ (q_i, q_j) \in T_0 = \{(k, l) \mid k \neq l, 1 \leq k, l \leq n, \}$
- $\forall 1 \leq d \leq n-1,\ \forall 1 \leq i \leq n-d, (q_i, q_{i+d}) \in T_d$
  with $T_d = \{(i, j) \mid \forall 1 \leq i \leq n, j \neq i + d \wedge j \neq i - d \wedge 1 \leq j \leq n\}$.

Note also that T_0 could be considered into the T_d tables (just adding $i \neq j$ in the definition of the set). However, separating the tables makes the "semantics" of each table clearer.

For the sake of simplicity and clarity, we now consider the 4-queens instance, which allows us to present the complete penalty matrices. Each q_i is represented by 4 Boolean variables $q_{i,1}, \ldots, q_{i,4}$ such that $q_{i,j} = 1$ iff $q_i = j$, $q_{i,j} = 0$ otherwise. A one-hot constraint, specifying that a variable has one and only one value is added for each i: $\sum_{j=1}^{4} q_{i,j} = 1$. This results in the following penalty submatrix:

$$m_{oh} = \begin{array}{c} \\ q_{i,1} \\ q_{i,2} \\ q_{i,3} \\ q_{i,4} \end{array} \begin{array}{c} \begin{array}{cccc} q_{i,1} & q_{i,2} & q_{i,3} & q_{i,4} \end{array} \\ \begin{pmatrix} -1 & 1 & 1 & 1 \\ 1 & -1 & 1 & 1 \\ 1 & 1 & -1 & 1 \\ 1 & 1 & 1 & -1 \end{pmatrix} \end{array}$$

The non-membership to T_0, T_1, T_2, and T_3 must be penalized. Following the automated process described in Section 4.2, we obtain the penalty sub-matrices showed in Figure 1. The global penalty matrix is thus composed of sums of these sub-matrices:

$$\begin{array}{c} \\ q_1 \\ q_2 \\ q_3 \\ q_4 \end{array} \begin{array}{c} \begin{array}{cccc} q_1 & q_2 & q_3 & q_4 \end{array} \\ \begin{pmatrix} m_{oh} & m_1 + m_0 & m_2 + m_0 & m_3 + m_0 \\ (m_1 + m_0)^\intercal & m_{oh} & m_1 + m_0 & m_2 + m_0 \\ (m_2 + m_0)^\intercal & (m_1 + m_0)^\intercal & m_{oh} & m_1 + m_0 \\ (m_3 + m_0)^\intercal & (m_2 + m_0)^\intercal & (m_1 + m_0)^\intercal & m_{oh} \end{pmatrix} \end{array}$$

It is obvious that the previous penalty matrices for 4 variables can be extended by expanding the pattern to modelize any n-queen instance.

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29918-5_1

$\forall i < j$,

$$m_0 = \begin{array}{c} \\ q_{j,1} \\ q_{j,2} \\ q_{j,3} \\ q_{j,4} \end{array} \begin{array}{c} \begin{array}{cccc} q_{i,1} & q_{i,2} & q_{i,3} & q_{i,4} \end{array} \\ \begin{pmatrix} 0 & -1 & -1 & -1 \\ -1 & 0 & -1 & -1 \\ -1 & -1 & 0 & -1 \\ -1 & -1 & -1 & 0 \end{pmatrix} \end{array}$$

$\forall 1 \leq i \leq n-1$,

$$m_1 = \begin{array}{c} \\ q_{i+1,1} \\ q_{i+1,2} \\ q_{i+1,3} \\ q_{i+1,4} \end{array} \begin{array}{c} \begin{array}{cccc} q_{i,1} & q_{i,2} & q_{i,3} & q_{i,4} \end{array} \\ \begin{pmatrix} -1 & 0 & -1 & -1 \\ 0 & -1 & 0 & -1 \\ -1 & 0 & -1 & 0 \\ -1 & -1 & 0 & -1 \end{pmatrix} \end{array}$$

$\forall 1 \leq i \leq n-2$,

$$m_2 = \begin{array}{c} \\ q_{i+2,1} \\ q_{i+2,2} \\ q_{i+2,3} \\ q_{i+2,4} \end{array} \begin{array}{c} \begin{array}{cccc} q_{i,1} & q_{i,2} & q_{i,3} & q_{i,4} \end{array} \\ \begin{pmatrix} -1 & -1 & 0 & -1 \\ -1 & -1 & -1 & 0 \\ 0 & -1 & -1 & -1 \\ -1 & 0 & -1 & -1 \end{pmatrix} \end{array}$$

$i = 1$,

$$m_3 = \begin{array}{c} \\ q_{i+3,1} \\ q_{i+3,2} \\ q_{i+3,3} \\ q_{i+3,4} \end{array} \begin{array}{c} \begin{array}{cccc} q_{i,1} & q_{i,2} & q_{i,3} & q_{i,4} \end{array} \\ \begin{pmatrix} -1 & -1 & -1 & 0 \\ -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 \\ 0 & -1 & -1 & -1 \end{pmatrix} \end{array}$$

Fig. 1. Penalty sub-matrices for the table constraints of the n-queen problem.

5.2 The *Traffic Lights* problem

Problem 16 from the CSPlib [1]: *Consider a four way traffic junction with 8 traffic lights. Four of the traffic lights are for the vehicles and can be represented by the variables v_1 to v_4 with domains $\{r, ry, g, y\}$ (for red, red-yellow, green and yellow). The other four traffic lights are for the pedestrians and can be represented by the variables p_1 to p_4 with domains $\{r, g\}$. The constraints on these variables can be modelled by quaternary constraints on (v_i, p_i, v_j, p_j) for $0 \leq i \leq 3,\ j = (1 + i) \bmod 4$ which allow just the tuples $\{(r, r, g, g), (ry, r, y, r), (g, g, r, r), (y, r, ry, r)\}$. We are interested in the set of all globally consistent 8-tuples (which reflects the evolution of the traffic light sequence).*

In CSP: 4 domain variables v_0, v_1, v_2, v_3 are required, each with a domain $\{1, 2, 3, 4\}$ encoding respectively $\{r, ry, g, y\}$, and 4 variables p_0, p_1, p_2, p_3 with a domain $\{1, 2\}$ encoding $\{r, g\}$. The allowed tuples are collected in a table: $T = \{(1, 1, 3, 2), (2, 1, 4, 1), (3, 3, 1, 1), (4, 1, 2, 1)\}$, and the constraints are:

$$(v_i, p_i, v_j, p_j) \in T \text{ for } 0 \leq i \leq 3,\ j = (1 + i) \mod 4 \tag{1}$$

In QUBO:

- each v_i is encoded by 4 Boolean variables $v_{i,j}$ such that $v_i = j$ iff $v_{i,j} = 1$
- each p_i is encoded by 2 Boolean variables $p_{i,j}$ such that $p_i = j$ iff $p_{i,j} = 1$
- one one-hot constraint for each v_i: $\sum_{j=1}^{4} v_{i,j} = 1$. The resulting penalty is:

$$\sum_{i=0}^{3} \sum_{\substack{j \neq j' \\ 1 \leq j, j' \leq 4}} v_{i,j} v_{i,j'} - \sum_{i=0}^{3} \sum_{j=1}^{4} v_{i,j}$$

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29918-5_1

– one one-hot constraint for each p_i: $\sum_{j=1}^{2} p_{i,j} = 1$ and the penalty:

$$\sum_{i=0}^{3} \sum_{\substack{j \neq j' \\ 1 \leq j,j' \leq 2}} p_{i,j} p_{i,j'} - \sum_{i=0}^{3} \sum_{j=1}^{2} p_{i,j}$$

The tuples of T are completed with a counter from 1 to 4. This builds the table T' which is then split into 4 tables T_i of couples (1 element of the initial tuple, and 1 number):

$$T_1 = \{(1,1),(2,2),(3,3),(4,4)\} \qquad T_2 = \{(2,1),(1,2),(2,3),(1,4)\}$$
$$T_3 = \{(3,1),(4,2),(1,3),(2,4)\} \qquad T_4 = \{(2,1),(1,2),(1,3),(1,4)\}$$

We need 4 ancillary variables c_1, c_2, c_3, c_4 for each of the 4 table constraints "$\in$". By splitting T and the tuples, we obtain 16 "$\in$" constraints:

$$\forall i \in [0..3],\ (v_i, c_{i+1}) \in T_1, \quad (p_i, c_{i+1}) \in T_2,$$
$$(v_{(i+1) \bmod 4}, c_{i+1}) \in T_3, \quad (p_{(i+1) \bmod 4}, c_{i+1}) \in T_4$$

Each c_i is represented by 4 Boolean variables $c_{i,j}$ with the one-hot constraint for each i: $\sum_{j=1}^{4} c_{i,j} = 1$, and the corresponding penalty

$$\sum_{i=1}^{4} \sum_{\substack{j \neq j' \\ 1 \leq j,j' \leq 4}} c_{i,j} c_{i,j'} - \sum_{i=1}^{4} \sum_{j=1}^{4} c_{i,j}$$

Finally, following the process of Section 4.3, each of the 8 constraints $(v_i, c_j) \in T_k$ and of the 8 constraints $(p_i, c_j) \in T_k$ is converted into its QUBO form:

$$\sum_{(l,m) \in T_k} v_{i,l} c_{j,m} \geq 1 \quad \text{and} \quad \sum_{(l,m) \in T_k} p_{i,l} c_{j,m} \geq 1 \tag{2}$$

And finally, the penalties for the 16 constraints $(v_i, c_j) \in T_k$ or $(p_i, c_j) \in T_k$ are:

$$\sum_{(l,m) \in T_k} -v_{i,l} c_{j,m} \quad \text{and} \quad \sum_{(l,m) \in T_k} -p_{i,l} c_{j,m} \tag{3}$$

6 The Costas Arrays Problem

With many applications for in remote sensing (e.g., RADAR and SONAR) and telecommunications, the Costas Array Problem (CAP) is a hard combinatorial problem that has been introduced in 1965 by John P. Costas [13]. A Costas array of size n is an $n \times n$ grid containing n marks such that there is exactly one mark per row and per column and the $n(n-1)/2$ vectors joining the marks are all different. More precisely, consider a first vector composed of two marks (i, j) and (k, l) and a second one composed of two marks (i', j') and (k', l'), then

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29918-5_1

they should not have the same difference between horizontal and vertical indices except if they are equal:

$$(k-i = k'-i' \ \wedge \ l-j = l'-j') \implies (i = i' \ \wedge \ j = j' \ \wedge \ k = k' \ \wedge \ l = l')$$

Although several methods have been designed for constructing Costas Arrays, e.g. Golomb or Welch methods, there is no general method to construct all Costas Arrays for any given size n, which is an NP-complete problem [5]. Thus one has to resort to exhaustive search for finding all Costas Arrays of size n. All Costas Arrays up to size 29 have been discovered, some for size 30 and 31, but the existence of a Costas Array of size 32 is still an open problem.

A direct modeling of the CAP with Boolean variables has been proposed in [10], which consists in considering n^2 Boolean variables x_{ij} equal to 1 if there is a mark on the cell (i, j) and to 0 otherwise, for $1 \leq i, j \leq n$. The objective function is composed of two parts, $H = H_v + H_p$, with:

$$H_v = \sum_{\substack{1\leq i<i'<n,\ j\in\{1,\cdots,n\},\ j'\in\{1,\cdots,n\} \\ 1<i+k\leq n-2,\ 1\leq j+l\leq n,\ 2<i'+k\leq n,\ 1<j'+l\leq n\}}} x_{ij}\ x_{i+k,j+l}\ x_{i'j'}\ x_{i'+k,j'+l}$$

and

$$H_p = -\sum_{i=1,j=1}^{n} x_{ij} + \sum_{k=1}^{n}\sum_{i<j} x_{ki}x_{kj} + \sum_{k=1}^{n}\sum_{i<j} x_{ik}x_{jk}$$

This PUBO/HOBO objective function has $-n$ as lower bound and the Costas Array property corresponds to the fact that this minimal value is actually achieved. Indeed, H_v is minimal when no two vectors (pair of 2 marks on the grid) are equal, and H_p is minimal there is a single mark on each line and each column, letting therefore the n marks represent a permutation of $\{1, \ldots, n\}$ with an adequate encoding. The quartic monomials in H_v can be quadratized one by one and produce $\mathcal{O}(n^6)$ additional Boolean variables (one for each monomial) or, more compactly, by the pairwise method [2], which will produce $\mathcal{O}(n^4)$ additional Boolean variables [10].

Let us now consider a CSP model with table constraints for the CAP. This model consists in a set of integer variables x_i and y_i, $i \in [1..n]$, each variable ranging from 1 to n, and a table $\bar{V}$ representing all forbidden relative positions of 4 marks (pairs of vectors) represented by $(i, j, i+k, j+l, i', j', i'+k, j'+l)$. $\bar{V}$ is thus defined by:

$$\begin{aligned}\bar{V} = &\{(i,j,i+k,j+l,i',j',i'+k,j'+l) \mid 1 \leq i < i' < n \ \wedge \ j,j' \in \{1,\cdots,n\} \\ &\wedge \ 2 < i+k \leq n \ \wedge \ 1 \leq j+l \leq n \ \wedge \ 2 < i'+k \leq n \ \wedge \ 1 < j'+l \leq n\}\end{aligned}$$

or equivalently and more explicitly:

$$\begin{aligned}\bar{V} = &\{(i_1,j_1,i_2,j_2,i_3,j_3,i_3+i_2-i_1,j_3+j_2-j_1) \\ &\mid 1 \leq i_1 < i_2 \leq n \ \wedge \ 1 \leq i_1 \leq i_3 \leq n \ \wedge \ j_1,j_2,j_3 \in \{1,\cdots,n\}\}\end{aligned}$$

The first formulation of $\bar{V}$ is more precise, while the second one is simpler.

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29918-5_1

Since we do not want pairs of vectors to be elements of the table $\bar{V}$, we consider non-element constraints linking 2 vectors (x_1, y_1, x_2, y_2) and (x_3, y_3, x_4, y_4) as described in the last paragraph of Section 4.3:

$$(x_1, y_1, x_2, y_2, x_3, y_3, x_4, y_4) \notin \bar{V} \iff (x_1, y_1, x_2, y_2, x_3, y_3, x_4, y_4) \in V$$

We can then derive the QUBO penalty from the table constraint with the scheme described in Section 4.3 for representing H_v, with one extra Boolean variable for each entry in the table. As there are $\mathcal{O}(n^8)$ tuples in the table V, this will introduce $\mathcal{O}(n^8)$ extra Boolean variables in total.

Note that a direct representation of the table $\bar{V}$ (with a direct QUBO penalty for elements of $\bar{V}$) would create only $\mathcal{O}(n^6)$ extra Boolean variables in total. Furthermore, by introducing pairwise covers [2] as an intermediate step between the original table and the derivation of the penalty, one could achieve a more compact model with $\mathcal{O}(n^4)$ extra Boolean variables [10].

We plan as future work to devise a general scheme for such optimizations in the transformation from the model with table constraints to the QUBO model.

7 Conclusion

In this paper, we have introduced a scheme for deriving a QUBO formulation for extensional constraints, specifically table constraints. This contribution fills a missing step toward a fully automated conversion process from CSP to QUBO. The resulting method is straightforward and computationally inexpensive. In particular, no auxiliary variables are required when handling binary tables. For n-ary table constraints, l Boolean variables are needed for a table containing l tuples, regardless of the arity n of the constraint. It is also worth noting that we generalize, systemize and formalize the ad-hoc conversion described in [12] for the special case of pairs of labels for the cyclic bandwidth problem. As future work, we plan to refine this framework in order to be able to use the notion of pairwise cover in a general manner for tables and thus automatically reduce the number of ancillary variables.

References

1. CSPLib: A problem library for constraints. https://www.csplib.org/
2. Anthony, M., Boros, E., Crama, Y., Gruber, A.: Quadratic reformulations of nonlinear binary optimization problems. Math. Program. **162**(1-2), 115–144 (2017)
3. Apt, K.R.: Principles of Constraint Programming. Cambridge University Press (2003)
4. Apt, K.R., Monfroy, É.: Constraint programming viewed as rule-based programming. Theory Pract. Log. Program. **1**(6), 713–750 (2001)
5. Barker, L., Drakakis, K., Rickard, S.T.: On the complexity of the verification of the costas property. Proc. IEEE **97**(3), 586–593 (2009)
6. Bessière, C., Régin, J.: Arc consistency for general constraint networks: Preliminary results. In: Proc. of the 15th Int. Joint Conference on Artificial Intelligence, IJCAI 97, Nagoya, Japan, 1997, 2 Volumes. pp. 398–404. Morgan Kaufmann (1997)

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29918-5_1

7. Boros, E., Hammer, P.L.: Pseudo-boolean optimization. Discrete Applied Mathematics **123**(1), 155–225 (2002)
8. Chancellor, N.: Domain wall encoding of discrete variables for quantum annealing and qaoa. Quantum Science and Technology **4**(4), 045004 (aug 2019)
9. Codognet, P.: Domain-wall / unary encoding in QUBO for permutation problems. In: IEEE International Conference on Quantum Computing and Engineering, QCE 2022, Broomfield, CO, USA, September 18-23, 2022. pp. 167–173. IEEE (2022)
10. Codognet, P.: Modeling the costas array problem in QUBO for quantum annealing. In: proceedings of EvoCOP 2022, 22nd European Conference on Evolutionary Computation in Combinatorial Optimization, Madrid, Spain. Lecture Notes in Computer Science, vol. 13222, pp. 143–158. Springer (2022)
11. Codognet, P.: Comparing QUBO models for quantum annealing: integer encodings for permutation problems. Int. Trans. Oper. Res. **32**(1), 18–37 (2025)
12. Codognet, P., Monfroy, É.: Modeling the cyclic bandwidth problem in QUBO for quantum annealing. In: Computational Science - ICCS 2025 Workshops - 25th International Conference, Singapore, Singapore, July 7-9, 2025, Proceedings, Part V. Lecture Notes in Computer Science, vol. 15911, pp. 135–149. Springer (2025)
13. Costas, J.: A study of detection waveforms having nearly ideal range-doppler ambiguity properties. Proceedings of the IEEE **72**(8), 996–1009 (1984)
14. Farhi, E., Goldstone, J., Gutmann, S., Sipser, M.: Quantum computation by adiabatic evolution (2000), `https://arxiv.org/abs/quant-ph/0001106`
15. Glover, F.W., Kochenberger, G.A., Hennig, R., Du, Y.: Quantum bridge analytics I: a tutorial on formulating and using QUBO models. Ann. Oper. Res. **314**(1), 141–183 (2022). https://doi.org/10.1007/S10479-022-04634-2
16. Hadfield, S., Wang, Z., O'Gorman, B., Rieffel, E.G., Venturelli, D., Biswas, R.: From the quantum approximate optimization algorithm to a quantum alternating operator ansatz. Algorithms **12**(2), 34 (2019). https://doi.org/10.3390/A12020034
17. Hammer, P.L., Rudeanu, S.: Boolean Methods in Operations Research and Related Areas. Springer (1968)
18. Ishikawa, H.: Transformation of general binary mrf minimization to the first-order case. IEEE Transactions on Pattern Analysis and Machine Intelligence **33**(6), 1234–1249 (2011)
19. Kadowaki, T., Nishimori, H.: Quantum annealing in the transverse Ising model. Physical Review E **58**, 5355–5363 (Nov 1998)
20. 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. Journal of Combinatorial Optimization **28**(1), 58–81 (2014)
21. Lecoutre, C., Likitvivatanavong, C., Yap, R.H.C.: A path-optimal GAC algorithm for table constraints. In: ECAI 2012 - 20th European Conference on AI. Montpellier, France, August 27-31 , 2012. Frontiers in AI and Applications, vol. 242, pp. 510–515. IOS Press (2012). https://doi.org/10.3233/978-1-61499-098-7-510
22. Lecoutre, C., Szymanek, R.: Generalized arc consistency for positive table constraints. In: Proc. of the 12th Int. Conference Principles and Practice of Constraint Programming - CP 2006, Nantes, France, 2006. Lecture Notes in Computer Science, vol. 4204, pp. 284–298. Springer (2006). https://doi.org/10.1007/11889205"˙22
23. Lucas, A.: Ising formulations of many NP problems. Frontiers in Physics **2** (2014)
24. Punnen, A.P.: The Quadratic Unconstrained Binary Optimization Problem: Theory, Algorithms, and Applications. Springer International Publishing (2022)
25. Richoux, F., Baffier, J., Codognet, P.: Learning QUBO models for quantum annealing: A constraint-based approach. In: Computational Science - ICCS 2023 - 23rd

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29918-5_1

International Conference, Prague, Czech Republic, July 3-5, 2023, Proceedings, Part V. LNCS, vol. 14077, pp. 153–167. Springer (2023)
26. Rodríguez-Heck, E.: Linear and quadratic reformulations of nonlinear optimization problems in binary variables. Ph.D. thesis, University of Liège, Belgium (8 2018)
27. Rosenberg, I.G.: Reduction of bivalent maximization to the quadratic case. Cahiers du Centre d'Etudes de Recherche Operationnelle pp. 71—74 (1975)
28. Rossi, F., van Beek, P., Walsh, T. (eds.): Handbook of Constraint Programming, Foundations of Artificial Intelligence, vol. 2. Elsevier (2006)
29. Samaras, N., Stergiou, K.: Binary encodings of non-binary constraint satisfaction problems: Algorithms and experimental results. Journal of Artificial Intelligence Research **24**, 641–684 (2005), `https://arxiv.org/pdf/1109.5714`
30. Tasseff, B., Albash, T., Morrell, Z., Vuffray, M., Lokhov, A., Misra, S., Coffrin, C.: On the emerging potential of quantum annealing hardware for combinatorial optimization. Journal of Heuristics **30**, 325–358 (2024)
31. Verma, A., Lewis, M.: Penalty and partitioning techniques to improve performance of qubo solvers. Discrete Optimization **44**, 100594 (2022)
32. Wang, R., Yap, R.H.C.: Bipartite encoding: A new binary encoding for solving non-binary csps. In: Bessiere, C. (ed.) Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020. pp. 1184–1191 (2020)
33. Wang, R., Yap, R.H.C.: The expressive power of ad-hoc constraints for modelling CSPs. In: 37th AAAI Conference on AI, AAAI 2023, 35th Conference on Innovative Applications of AI, IAAI 2023, 13th Symposium on Educational Advances in AI, EAAI 2023, Washington, DC, USA, 2023. pp. 4104–4114. AAAI Press (2023)

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29918-5_1