



HAL
open science

Quadratic versus Polynomial Unconstrained Binary Models for Quantum Optimization illustrated on Railway Timetabling

Camille Grange, Marion Lavignac, Valentina Pozzoli, Eric Bourreau

► **To cite this version:**

Camille Grange, Marion Lavignac, Valentina Pozzoli, Eric Bourreau. Quadratic versus Polynomial Unconstrained Binary Models for Quantum Optimization illustrated on Railway Timetabling. 2024. hal-04783464

HAL Id: hal-04783464

<https://hal.science/hal-04783464v1>

Preprint submitted on 14 Nov 2024

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Quadratic versus Polynomial Unconstrained Binary Models for Quantum Optimization illustrated on Railway Timetabling

Camille Grange^{1,2,3}, Marion Lavignac^{3,4}, Valentina Pozzoli², and Eric Bourreau¹

¹LITIS, Normandie Univ, UNIHAVRE, 25 rue Philippe Lebon, Le Havre, France, email:

camille.grange@univ-lehavre.fr

²LIRMM, Univ Montpellier, CNRS, 161 rue Ada, Montpellier, France, email: eric.bourreau@lirmm.fr

³SNCF, Technology, Innovation and Group Projects Department, 1 avenue François Mitterand,

Saint-Denis, France, email: valentina.pozzoli@sncf.fr

⁴CentraleSupélec, Paris-Saclay University, 3 rue Joliot-Curie, Gif-sur-Yvette, France, email:

marion.lavignac@student-cs.fr

Abstract

Quantum Approximate Optimization Algorithm (QAOA) is one of the most short-term promising quantum-classical algorithm to solve unconstrained combinatorial optimization problems. It alternates between the execution of a parametrized quantum circuit and a classical optimization. There are numerous levers for enhancing QAOA performances, such as the choice of quantum circuit meta-parameters or the choice of the classical optimizer. In this paper, we stress on the importance of the input problem formulation by illustrating it with the resolution of an industrial railway timetabling problem. Specifically, we present a generic method to reformulate any polynomial problem into a Polynomial Unconstrained Binary Optimization (PUBO) problem, with a specific formulation imposing penalty terms to take binary values when the constraints are linear. We also provide a generic reformulation into a Quadratic Unconstrained Binary Optimization (QUBO) problem. We then conduct a numerical comparison between the PUBO with binary penalty terms and the QUBO formulations proposed on a railway timetabling problem solved with QAOA. Our results illustrate that the PUBO reformulation outperforms the QUBO one for the problem at hand.

keywords: Combinatorial optimization, Quantum optimization, QAOA, Railway timetable, Unconstrained binary model

1 Introduction

Today, a major part of optimization problems in the industrial context are challenging, with resolutions that remain difficult using current classical methods. The difficulty in solving these problems, namely finding the optimal solution(s), stems both from the inherent combinatorial complexity, often NP-hard, and from the industrial scale of the instances involved. Facing these obstacles, quantum computing is expected to both improve solution quality and reduce computation time. Such expectations are not demonstrated on problems of industrial, practical scale yet, and represent ongoing research. Some operational search methods using quantum algorithms present theoretical advantages compared to classical ones (Ambainis et al., 2019; Grange et al., 2024; Montanaro, 2020; Nannicini, 2021; Kerenidis and Prakash, 2020), but they require high-quality quantum resources to be implemented. Specifically, they need quantum computers with many qubits that can interact two by two, and quantum operations that can be applied in a row without generating too much noise. However, the current quantum computers do not respect these criteria, encouraging the researchers to look into *lighter* algorithms, waiting for more powerful machines to be built. They consist of hybrid algorithms that take advantage of both quantum and classical resources. Indeed, quantum resources can focus on the critical aspects of the algorithm that utilize quantum information theory, while the more computationally demanding tasks are handled by classical resources. Such algorithms are metaheuristics, the most famous being the Quantum Approximate Optimization Algorithm (QAOA) introduced by Farhi et al. (2014) to solve the MAX-CUT problem. QAOA takes place in the larger class of Variational Quantum Algorithms (VQAs) (Cerezo et al., 2021; Grange et al., 2023) which consist of alternating between a quantum circuit and a classical optimizer. Even if these algorithms have no performance guarantee for general problems, they are of great interest today because they have the convenient property of an adjustable quantum circuits' depth, making them implementable on the current NISQ computers (Preskill, 2018).

Recently, several combinatorial problems have been solved using QAOA. One can mention *theoretical* problems such as MAX-CUT (Farhi et al., 2014), Travelling Salesman Problem (Ruan et al., 2020), MAX-3-SAT (Nannicini, 2019), Graph Coloring (Tabi et al., 2020), Job Shop Scheduling (Kurowski et al., 2023) and Vehicle Routing (Azad et al., 2022). More *industrial* problems have also been tackled, such as knapsack problem for battery revenue (de la Grand'rive and Hullo, 2019) or smart charging of electric vehicles (Dalyac et al., 2021; Kea et al., 2023). However, due to the small size of instances processed today (imposed by the weak maturity of quantum computers) and to the nature of heuristics whose performances are evaluated empirically, no quantum advantage is emerging yet.

Several levers are available when solving a problem with QAOA. For instance, one can choose the function that guides the classical optimizer, the depth or the type of gates used in the quantum circuit. Their impact in QAOA’s performances have been proved (Barkoutsos et al., 2020; Li et al., 2020; Nannicini, 2019). However, the lever of the formulation of the combinatorial optimization problem has been less tackled in the literature, even though the shape of the quantum circuit is known to depend directly on the input problem. VQAs, and thus QAOA, necessitate the combinatorial optimization problem to be formulated as an Unconstrained Binary Optimization one, where the function is polynomial. While many real-world problems are constrained, transforming them into unconstrained problems represents a significant challenge, which could improve the performance of QAOA. The impact of the reformulation was first studied for Quadratic Unconstrained Binary Optimization (QUBO) reformulations. Indeed, QAOA derives from the Quantum Annealing algorithm of Kadowaki and Nishimori (1998), algorithm that takes as input an Ising Model, a model of ferromagnetism in statistical mechanics, which is equivalent to a QUBO problem. Thus, natively, the comparison of different formulations of QUBO models for a given problem recently raised interest in the community, for instance for the graph coloring problem (Tabi et al., 2020) or for Max-k-colorable subgraph problem (Quintero et al., 2022). However, there are in fact no inherent degree limitations of the objective function. Some studies investigate Polynomial Unconstrained Binary Optimization (PUBO) formulations but, as far as we know, a generic reformulation approach is still lacking. These studies compare QAOA’s performances of PUBO and QUBO formulations for specific problems such as for the Traveling Salesman Problem (Salehi et al., 2022), the graph coloring problem (Campbell and Dahl, 2022), or a minimization problem with continuous variables (Stein et al., 2023). However, they show an advantage of PUBO formulations that is often attributed to the reduction of the number of qubits required, though this may not be the only factor at stake. PUBO formulations offer several benefits: they not only reduce the number of qubits significantly but they also provide greater flexibility in shaping constraints, for instance by assigning specific binary values to the penalty functions.

In this paper, we propose a generic reformulation for any polynomial problem into a PUBO problem, alongside with a specific reformulation in the case of linear constraints, exploiting PUBO’s ability to enforce penalty terms to take 0 and 1 values. We also present two generic QUBO reformulations for comparison, also applied to polynomial problems. Finally, we numerically compare QAOA performances of the PUBO formulation with binary penalty values, with one of the QUBO formulations on an industrial use-case: a railway timetabling problem. The timetabling we study consists of finding the transportation plan maximizing the operat-

ing profit according to the customers’ demand taking into account the availability and cost of both the network and the rolling stock. It has been developed by the research department of the French Railway Company SNCF in order to help operators conceiving optimal high-speed train timetables. Its simplification, necessary because of the weak maturity of current quantum computers, leads to a Bin Packing problem with slight modifications on the constraints.

The paper is structured as follows. In Section 2, we propose generic reformulations of any problem with integer variables, polynomial objective function and polynomial constraints into PUBO and QUBO problems. We also provide a specific PUBO reformulation for the case of linear constraints, enforcing penalty terms to take binary values. In Section 3, we describe the nominal railway timetabling problem, we present a simplified version and we reformulate it as a PUBO and a QUBO following the methods of Section 2. Eventually, in Section 4, we illustrate the impact of the reformulation on QAOA’s performances by solving several instances, revealing that PUBO with binary penalty values performs better than QUBO for our problem.

2 Generic reformulations

Variational Quantum Algorithms (VQAs) tackle unconstrained problems. However, most real-world combinatorial problems are constrained. Some constraints are directly related to the definition of the problem, for instance, that a city is visited exactly once by the salesman in the TSP. Some others express real-world limits, such as the limited number of seats on a train or the finite size of a knapsack, involving numerical constants in the description of the problem. In both cases, we need to reformulate the problem as an unconstrained optimization problem to solve it with VQAs. To remove the constraints, a common approach is to integrate them as penalty terms in the objective function. Examples of such reformulations for several NP-hard problems into QUBO problems are proposed in the literature (Glover et al., 2022; Lucas, 2014). Notice that other techniques to remove constraints have been proposed, such as modifying the circuit of VQAs to *express* the constraints (Hadfield et al., 2019).

In this section, we aim to widen the range of problems we can address with VQAs. For that, we provide a general method to transform any problem with integer variables, a polynomial objective function, and polynomial constraints into an *equivalent* PUBO problem. Then, taking advantage of the wide possibilities of PUBO formulations, we propose a specific PUBO formulation with binary penalty terms, for problems with linear constraints only. Finally, we provide two different general transformations into an *equivalent* QUBO problem. Two problems are said to be equivalent if and only if they have the same optimal solution(s) with the same optimal value.

2.1 Integration of constraints

First, we present a generic way to remove constraints from a nominal constrained problem and ensure that the reformulation into the resulting unconstrained problem is valid, i.e. that the resulting problem is *equivalent* to the nominal problem. Specifically, let us consider the problem

$$\begin{aligned} \min_{x \in \mathcal{X}} \quad & f(x) \\ \text{subject to} \quad & \text{the set of constraints } \{k \in \mathcal{K}\} \end{aligned} \tag{Nominal}$$

For each constraint $k \in \mathcal{K}$, we define a penalty function $\pi_k : \mathcal{X} \rightarrow \mathbb{R}^+$ that satisfies, for $x \in \mathcal{X}$,

$$\pi_k(x) \begin{cases} = 0 & \text{if } x \text{ satisfies constraint } k \\ \geq 1 & \text{if } x \text{ violates constraint } k \end{cases}$$

We reformulate the constrained problem (Nominal) as follows.

Definition 2.1. *The unconstrained problem, for which we integrate the constraints of the nominal problem as penalty terms in the objective function, is*

$$\min_{x \in \mathcal{X}} f(x) + \sum_{k \in \mathcal{K}} \lambda_k \pi_k(x), \tag{Unconstrained}$$

where the $\lambda_k > 0$ are penalty coefficients.

Thus, the reformulation of (Nominal) into (Unconstrained) not only requires finding the penalty functions π_k , as we will discuss in Subsections 2.3 and 2.4, but also requires choosing the numerical values of the penalty coefficients λ_k . Next, we provide a general lower bound for them.

Let us note $f_{\min} := \min\{f(x) : x \in \mathcal{X}\}$ the minimum value of f , $f_{\max} := \max\{f(x) : x \in \mathcal{X}\}$ its maximum value, and $f^* := \min_{x \in \mathcal{X}}\{f(x) : x \text{ respects constraint } k, \forall k \in \mathcal{K}\}$ the optimal value of (Nominal).

Proposition 2.2. *If we set, for all $k \in \mathcal{K}$,*

$$\lambda_k \geq f_{\max} - f_{\min},$$

thus, we ensure that (Nominal) is equivalent to (Unconstrained).

Proof. The smallest value of an unfeasible solution of (Unconstrained) is always larger than the minimum value of f plus the penalty cost of violating at least one constraint k' , namely, larger than

$$f_{\min} + \lambda_{k'}.$$

Thus, for $\lambda_k \geq f_{\max} - f_{\min}$ for any constraint k , the smallest value of an unfeasible solution

of (Unconstrained) is larger than

$$f_{\min} + \lambda_{k'} \geq f_{\min} + f_{\max} - f_{\min} = f_{\max} > f^*.$$

Moreover, by definition of the penalty function, the values of the objective function of the two problems coincide on feasible solutions. Consequently, the optimal value of (Nominal), and its corresponding solution, is equal to the one of (Unconstrained). \square

Notice that if we set $\lambda_k \geq f_{\max} - f_{\min}$ for all $k \in \mathcal{K}$, not only (Unconstrained) has the same optimal solution as (Nominal), but also any feasible solution (of (Nominal)) has a lower loss function value in (Unconstrained) than any unfeasible solution (of (Nominal)). This latter property is worth noting because VQAs are heuristics, so they do not always find the optimal solution but solutions close to the optimal (in terms of the loss function). In practice, if $f_{\min} \geq 0$, an upper bound of f_{\max} provides a lower bound for each λ_k . Thus, setting $\lambda_k \geq f_{\max}$ ensures the new unconstrained problem to satisfy the above-mentioned properties.

In this subsection, we showed that reformulating a constrained problem into an unconstrained problem amounts to finding the penalty functions for each constraint. Next, we present a broad class of problems for which we provide the expression of the penalty functions.

2.2 Class of eligible problems

We present a class of problems (IP-poly) for which we provide next a method to reformulate them as PUBO and QUBO problems. This class contains problems with integer variables, a polynomial objective function, and polynomial constraints.

Definition 2.3. *Let $n \in \mathbb{N}$ be the number of variables and let $m \in \mathbb{N}$ be the number of constraints. We call (IP-poly) the following class of problems.*

$$\begin{array}{lll} \min_x & f(x_1, \dots, x_n) & \text{(IP-poly)} \\ \text{subject to} & g_k(x_1, \dots, x_n) \leq 0, & \forall k \in [m] \\ & x_i \in \mathbb{N}, & \forall i \in [n] \end{array}$$

where the functions f and g_k , for any $k \in [m] := \{1, \dots, m\}$, are polynomial. Specifically,

$$\begin{array}{lll} \min_x & \sum_{\gamma=(\gamma_1, \dots, \gamma_n) \in \Gamma} \alpha_\gamma x_1^{\gamma_1} \dots x_n^{\gamma_n} & \text{(IP-poly)} \\ \text{subject to} & \sum_{\gamma=(\gamma_1, \dots, \gamma_n) \in \Gamma_k} \beta_{k,\gamma} x_1^{\gamma_1} \dots x_n^{\gamma_n} \leq 0, & \forall k \in [m] \quad (\text{C}_k) \\ & x_i \in \mathbb{N}, & \forall i \in [n] \end{array}$$

where $\Gamma \subseteq \mathbb{N}^n$ is a finite set and $\alpha_\gamma \in \mathbb{R}$ for $\gamma \in \Gamma$; and for all $k \in [m]$, $\Gamma_k \subseteq \mathbb{N}^n$ is a finite set and $\beta_{k,\gamma} \in \mathbb{Z}$ for $\gamma \in \Gamma_k$.

2.3 Reformulation into PUBO

In this subsection, we present a generic method to reformulate any problem of (IP-poly) into a PUBO problem (Subsubsection 2.3.1). We also provide a reformulation in the case of linear constraints of the nominal problem (Subsubsection 2.3.2), enforcing a certain shape of the penalty functions.

2.3.1 General PUBO reformulation

The two steps to reformulate any problem of (IP-poly) into a PUBO problem are the following. First, we transform the integer variables into binary variables. Second, we integrate the constraints into the objective function as penalty terms. Let us specify each of these steps.

The first step is the transformation of integer variables into binary variables. For that, we replace each integer variable x_i , for $i \in [n]$, by its binary decomposition

$$x_i = \sum_{j=0}^{\lfloor \log_2(x) \rfloor} x_i^{(j)} 2^j.$$

Thus, this decomposition requires $\lfloor \log_2(x) \rfloor + 1$ binary variables $x_i^{(j)} \in \{0, 1\}$.

The second step is the integration of the constraints into the objective function as penalty terms. It is done as follows. Let us consider the constraint (C_k)

$$\sum_{\gamma=(\gamma_1, \dots, \gamma_n) \in \Gamma_k} \beta_{k,\gamma} x_1^{\gamma_1} \dots x_n^{\gamma_n} \leq 0. \quad (C_k)$$

This step aims at finding a penalty function for the above-mentioned constraint. Notice that after the first step, all variables are binary, so the integer variables of the left-hand side of (C_k) would be replaced by their binary description, adding more terms to the sum. Thus, without loss of generality, we assume henceforth that all the x_i are binary variables. It results the following upper bound:

$$\left| \sum_{\gamma \in \Gamma_k} \beta_{k,\gamma} x_1^{\gamma_1} \dots x_n^{\gamma_n} \right| \leq \sum_{\gamma \in \Gamma_k} |\beta_{k,\gamma}| =: \text{UB}_k.$$

Thus, we define the penalty function associated with Constraint (C_k) as follows.

Proposition 2.4. *For $k \in [m]$, the function*

$$\pi_k^{PUBO}(x_1, \dots, x_n) = \prod_{j=0}^{\text{UB}_k} \left(\sum_{\gamma \in \Gamma_k} \beta_{k,\gamma} x_1^{\gamma_1} \dots x_n^{\gamma_n} + j \right)$$

is a penalty function for Constraint (C_k) .

Proof. On the one hand, if (x_1, \dots, x_n) satisfies the constraint, it means that $\sum_{\gamma \in \Gamma_k} \beta_{k,\gamma} x_1^{\gamma_1} \dots x_n^{\gamma_n}$ takes a value in $[-UB_k, 0]$ and then there exists $j \in [0, UB_k]$ that makes the product equal to 0, i.e. $\pi_k^{\text{PUBO}}(x_1, \dots, x_n) = 0$. On the other hand, if (x_1, \dots, x_n) violates the constraint, the term $\sum_{\gamma \in \Gamma_k} \beta_{k,\gamma} x_1^{\gamma_1} \dots x_n^{\gamma_n}$ is strictly positive. Precisely, because each $\beta_{k,\gamma}$ is in \mathbb{Z} , the term $\sum_{\gamma \in \Gamma_k} \beta_{k,\gamma} x_1^{\gamma_1} \dots x_n^{\gamma_n}$ cannot be smaller than 1, leading to $\pi_k^{\text{PUBO}}(x_1, \dots, x_n) \geq 1$. \square

2.3.2 Case of linear constraints

Let us suppose that the constraints of (IP-poly) are linear. In that case, we propose penalty functions that take binary values. Specifically, the penalty function takes value 1 if the corresponding constraint is violated and 0 otherwise. The choice of such binary functions is motivated by understanding whether or not controlling the violation of cost of each constraint improves performances of VQAs to solve such problems. This is discussed in Section 4.

In what follows, we provide penalty functions of binary values of equality and inequality constraints where the variable-dependent term is the sum of binary variables. Notice that for the case of a *weighted* sum of binary variables (with weights in \mathbb{Z}), the penalty function can be easily deduced.

Let us begin with the penalty term for equality constraints.

Property 2.5. *Let us consider the constraint, for $n \in \mathbb{N}$ and $c \in \mathbb{N}$,*

$$\sum_{i=1}^n x_i = c. \quad (\text{Eq})$$

For $c \in \mathbb{N}^*$, the penalty function is, for $x \in \{0, 1\}^n$,

$$\pi_c^{\text{eq}}(x) = 1 + \sum_{k=c}^n (-1)^{k-c+1} \binom{k}{c} \sum_{i=\{i_1, \dots, i_k\} \in I_k^n} x_{i_1} \dots x_{i_k},$$

where I_k^n denotes all the sets of k elements in $[n]$. For the specific case of $c = 0$, the penalty function is

$$\pi_0^{\text{eq}}(x) = \sum_{k=1}^n (-1)^{k+1} \sum_{i \in I_k^n} x_{i_1} \dots x_{i_k}.$$

See proof in Appendix A.1. Next, we define a penalty term for inequality constraints. Property 2.6 deals with the inferiority case whereas Property 2.7 tackles the superiority case.

Property 2.6. *We consider the constraint, for $n \in \mathbb{N}$ and $c \in \mathbb{N}$,*

$$\sum_{i=1}^n x_i \leq c. \quad (\text{Inf})$$

The associated penalty function is, for $x \in \{0, 1\}^n$,

$$\pi_c^{inf}(x) = \sum_{k=c+1}^n (-1)^{k-c+1} \binom{k-1}{c} \sum_{i \in I_k^n} x_{i_1} \dots x_{i_k}.$$

See proof in Appendix A.2.

Property 2.7. We consider the constraint, for $n \in \mathbb{N}$ and $c \in \mathbb{N}^*$,

$$\sum_{i=1}^n x_i \geq c. \quad (\text{Sup})$$

The associated penalty function is, for $x \in \{0, 1\}^n$,

$$\pi_c^{sup}(x) = 1 + \sum_{k=c}^n (-1)^{k-c+1} \binom{k-1}{c-1} \sum_{i \in I_k^n} x_{i_1} \dots x_{i_k}.$$

See proof in Appendix A.3.

2.4 Reformulation into QUBO

Any problem of (IP-poly) can also be reformulated as a QUBO problem. We present two possible generic ways to do so.

The first one is to compute the PUBO penalty term π_k^{PUBO} resulting of the two steps presented in the previous subsection, and hereafter to decrease its degree until it is equal to 2. For that, we reduce recursively the degree of each monome of degree larger than 3 by using the following method of linearization. Repeat until no monome of degree at least 3 exists:

1. Choose a monome of degree larger than 3.
2. Pick two (binary) variables appearing in the monome x_i and x_j and replace the product $x_i x_j$ by the new binary variable y_{ij} in every monomes of the penalty function $\pi_k^{\text{PUBO}}(x)$ where it appears.
3. Add the term $\lambda(x_i x_j - 2x_i y_{ij} - 2x_j y_{ij} + 3y_{ij})$ to π_k^{PUBO} , with $\lambda > 0$ a constant.

Thus, it decreases the degree of the considered monome (and possibly other monomes) by 1. Notice that the term

$$pen_{\text{lin}}(x_i, x_j, y_{ij}) := x_i x_j - 2x_i y_{ij} - 2x_j y_{ij} + 3y_{ij}$$

is a penalty term associated to the constraint linearization $x_i x_j = y_{ij}$. Indeed, Table 1 shows that pen_{lin} is equal to zero if the value of the triplet (x_i, x_j, y_{ij}) satisfies the constraint (rows in bold), and larger than 1 otherwise.

The second way to provide quadratic penalty terms is the following. Given the constraint with binary variables (C_k) resulting from the first step of PUBO formulation presented in the

x_i	x_j	y_{ij}	$pen_{lin}(x_i, x_j, y_{ij})$
0	0	0	0
0	1	0	0
1	0	0	0
1	1	0	1
0	0	1	3
0	1	1	1
1	0	1	1
1	1	1	0

Table 1: Values of function pen_{lin}

previous subsection, linearize it as proposed above. Let us note $Lin_k(x, y) \leq 0$ the resulting constraint, where the y variables are resulting from the linearization process. Thus, the penalty term associated with this constraint is

$$\pi_k^{\text{QUBO}}(x, y) = \min_{s \in \llbracket 0, -\min_{x,y} Lin_k(x,y) \rrbracket} (Lin_k(x, y) + s)^2.$$

Indeed, for a given (x, y) :

- If the constraint is satisfied, i.e. $Lin_k(x, y) \leq 0$, there exists a value of $s \in \llbracket 0, -\min_{x,y} Lin_k(x, y) \rrbracket$ such that $Lin_k(x, y) + s = 0$, and thus $\pi_k^{\text{QUBO}}(x, y) = 0$.
- Otherwise, $Lin_k(x, y) > 0$ and there is no value in $s \in \llbracket 0, -\min_{x,y} Lin_k(x, y) \rrbracket$ such that $s = -Lin_k(x, y)$, and thus $\pi_k^{\text{QUBO}}(x, y) \geq 1$.

Notice that we suppose that the constraint can be satisfied, namely that $\min_{x,y} Lin_k(x, y) \leq 0$.

Eventually, because VQAs can only deal with one optimization problem, the optimization problems \min_s resulting from the expression of the penalty functions π_k^{QUBO} must join the initial optimization problem $\min_{x,y}$ of the loss function resulting of the first step of PUBO formulation. Gathering the optimization problems of penalty terms with the nominal optimization problem leads to a overall minimization problem over the slack variables (s) in addition to the decision variables (x, y) . The optimal solution(s) of the overall minimization problem coincide with the optimal solution(s) of the two-level optimization problem, but it is worth noting that the optimization process is different and leads, among other, to non-optimal values of slack variables for optimal values of decision variables. This phenomenon is discussed in Section 4.

Thus, there are m additional variables $s_k \in \llbracket 0, -\min_{x,y} Lin(x, y) \rrbracket$ (eventually written with binary variables), for $k \in [m]$, in the overall optimization QUBO problem. Henceforth, we call

these variables *slack* variables to emphasize the fact that they do not represent nominal decision variables but additional variables coming from the reformulation.

Next, we present the railway timetabling problem at hand and apply both QUBO and PUBO reformulations presented in the section.

3 A railway timetabling problem and its reformulations

3.1 Nominal problem

Railway timetabling problems are crucial problems for railway companies. A first version of the timetable is often planned several years in advance, and is related to other planning problems, such as crew scheduling or rolling stock scheduling. The goal of timetabling is to ensure the satisfaction of customers, the minimization of delays thanks to robustness and resilience properties, the minimization of costs, and the validation of both operational and security constraints.

The railway timetabling problem for high-speed trains at the French Railway Company SNCF we are considering is the following: according to the customers' demand (estimated from past data) and the availability of the network and the rolling stock, the aim is to find the transportation plan maximizing the operating profit. The output is a timetable of trains, the associated rolling stock schedule, and a forecast of the passengers for each train and journey. The optimal solution is the best compromise between the revenues generated by the customers' journeys and the production costs (network, rolling stock, human resources, etc.). It is formulated as an Integer Linear Programming (ILP). Let us provide a basic description of it while explaining the important ideas to understand the simplification proposed below. First, we describe the main sets of an instance of our problem.

- S , the set of Train-paths. A train-path is a timed unitary portion of tracks. It defines the availability to run a carriage over a portion of tracks over a given time period.
- T , the set of available Trains. A train is described as a union of train-paths. A train is defined by its origin, destination, and the served stations, with departure and arrival times for each station. A train uses the same carriage throughout the journey.
- G , the set of Groups of customers. A group gathers customers that have the same preferences on journeys, namely, customers wanting to leave, respectively arrive, at the same station and at the same time.
- $R = \{(t, g) \in T \times G : \text{group } g \text{ accepts to take train } t\}$, the set of possible Customers' journeys. This set expresses the possibilities to satisfy the customer groups' demands.

Other sets are required, such as the set expressing the different incompatibilities between train-paths or the set of trains that can be coupled. Besides sets, many constants appear in the original formulation, both in the objective function and in the constraints, such as the maximum capacity of a carriage, the toll cost of a train-path, or the average receipt for a journey.

Second, let us introduce the variables of this problem, which are binary or integers.

- $x \in \{0, 1\}^{|T|}$, where $x_t = 1$ iff $t \in T$ is used in the timetable.
- $u \in \{0, 1\}^{|S|}$, where $u_s = 1$ iff the train-path $s \in S$ is used.
- $z \in \mathbb{N}^{|R|}$, where z_r is the number of customers for journey $r \in R$.

The objective function leads to finding the timetable providing the best compromise between customers' demand satisfaction and production cost. Specifically,

$$\max_{x_t, u_s, z_r} \alpha(z_r) - \beta(x_t, u_s),$$

where $\alpha(z_r)$ is a linear function representing the receipts generated by selling tickets and $\beta(x_t, u_s)$ is a linear function representing the total cost associated with the use of train-paths and carriages. An optimal solution to our problem is a feasible timetable that maximizes the above loss function. The feasibility of a timetable is defined by many linear constraints such as forbidding customers to take a train not used in the timetable, ensuring that the maximum capacity of a train is satisfied on each train-path, or setting a minimum daily frequency for a given journey. In practice, this railway timetabling problem requires about twenty sets and constants to define an instance of the nominal problem, while it requires four types of binary or integer variables, a linear function, and ten types of linear constraints, two of which are soft constraints. Today, this problem is solved with an ILP solver and, depending on the instance, it can be hard or even impossible to find an optimal or near-optimal solution. For example, the optimal solution of the problem for the sector between two French cities Paris and Lyon is found within a second whereas the solution for timetabling of the inter-regional trains has a gap of 67% from the optimal solution after 10 minutes of running time. No solution is found for larger instances such as the perimeter of the entire metropolitan French territory.

While Section 2 proves that this railway timetabling problem can be solved theoretically with quantum-classical metaheuristics, because it belongs to (IP-poly), we need to simplify it by putting aside some assumptions to be able to solve it on current quantum machines, or classical simulators of quantum machines. Indeed, the size of the original problem would be too large even considering instances on small geographical sectors and short periods of time (one day). For example, the instance on the sector Paris-Lyon for one day only, which covers 6 stations,

amounts to 176 customer groups for around 25 000 customers, 87 feasible trains, and 157 train-paths leads to a nominal problem with 8 000 binary variables. Additionally, VQAs require unconstrained problems (QUBO or PUBO problems), integrating constraints in the objective function costs in terms of additional variables (for QUBO reformulation) and additional gates (for QUBO and PUBO reformulation) as explained in Section 2. As an example, the QUBO formulation of the above-mentioned instance requires roughly 12 000 additional qubits, ending up to 20 000 qubits for the total description of the instance. This size prohibits a resolution on current gate-based quantum hardware that does not exceed a hundred qubits with a reasonable gate error rate, without mentioning the high connectivity that would be necessary. While keeping the essence of the initial timetabling problem, we consider a simplification that is an extended version of a Bin Packing problem, as we detail next.

3.2 Extended Bin Packing Problem simplification

Let us present the simplified version of the nominal railway problem that results in an Extended Bin Packing problem. This simplification retains the core aspects of the original problem while relaxing certain operational requirements. As mentioned above, this simplification is motivated by two things. The first one is to reduce the number of decision variables, and thus the number of qubits, to describe an instance. The second one is to ease the transformation of the constrained problem into an unconstrained one, avoiding too many additional qubits and too many gates for the implementation of the quantum circuit of VQAs. Moreover, the Extended Bin Packing problem is NP-hard as the nominal problem, which comforts the interest of this choice.

The simplification is done as follows. Let $m \in \mathbb{N}$ be the number of customer groups and $n \in \mathbb{N}$ the number of available trains. We define

- $G := \{g_1, \dots, g_m\}$, a set of $m \in \mathbb{N}$ customer groups.
- $T := \{T_1, \dots, T_n\}$, the set of $n \in \mathbb{N}$ available trains, where each train $T_i \subseteq G$ is the set of groups that accept to take this train.

We consider that a customer group g_k , for $k \in [m]$, is satisfied if at least one of the trains matching its demand (the T_i s such that $g_k \in T_i$) is in the output timetable.

Additionally, we suppose that each group contains the same number of customers. The latter assumption does not cause too much loss of generality because one can always define the smallest group as a unit and duplicate groups that are bigger. For each available train T_i , $i \in [n]$, we specify

- $p_i \in \mathbb{R}^+$, the benefit of selling tickets to one group for train T_i .

- $c_i \in \mathbb{R}^+$, the cost of using train T_i .

We note $CMax \in \mathbb{N}$ the maximum number of groups a carriage can accommodate, i.e. the maximum capacity of a carriage divided by the (fixed) number of customers in a group. We define below the problem considered, called Extended Bin Packing.

Definition 3.1 (Extended Bin Packing problem). *The binary decision variables for the Extended Bin Packing problem are of two types. The first one indicates if the train is taken in the timetable: $\forall i \in [n]$,*

$$x_i = \begin{cases} 1 & \text{if } T_i \text{ is taken in the timetable} \\ 0 & \text{else} \end{cases}$$

The second assigns groups to trains in the timetable: $\forall i \in [n], \forall j \in [m]$,

$$y_{ij} = \begin{cases} 1 & \text{if group } g_j \text{ takes train } T_i \\ 0 & \text{else} \end{cases}$$

Notice that we declare a variable y_{ij} if and only if $g_j \in T_i$, namely that train T_i can satisfy group g_j . It avoids unnecessary variables and reduces the number of y variables from nm to $q := \sum_{i=1}^n |T_i|$. The problem can be seen as a variation of the Bin Packing problem, where each bin has a different capacity, and not every item has to be put in a bin because we are rather looking for the best compromise between the cost of using bins and the reward of putting items into bins. Hence we call this modelization the Extended Bin Packing problem. It is stated as follows:

$$\begin{aligned} \min_{x,y} \quad & \sum_{i=1}^n c_i x_i - \sum_{i=1}^n \sum_{\substack{j \in [m]: \\ g_j \in T_i}} p_i y_{ij} & & \text{(Extended-BP)} \\ \text{subject to} \quad & \sum_{\substack{i \in [n]: \\ g_j \in T_i}} y_{ij} \leq 1, & \forall j \in [m] & \text{(Uni)} \\ & \sum_{\substack{j \in [m]: \\ g_j \in T_i}} y_{ij} \leq CMax \cdot x_i, & \forall i \in [n] & \text{(Capa)} \\ & x_i \in \{0, 1\}, & \forall i \in [n] \\ & y_{ij} \in \{0, 1\}, & \forall i \in [n], j \in [m] \text{ such that } g_j \in T_i \end{aligned}$$

Constraint (Uni) ensures that each customer group takes at most one train from those available in the timetable, and Constraint (Capa) both expresses the limited capacity of a carriage and forbids a group to take a train unused in the timetable.

Next, we provide reformulations of this simplified model into QUBO and PUBO problems to be able to solve it with QAOA.

3.3 Reformulations into QUBO and PUBO problems

In this subsection, we provide reformulations of (Extended-BP) into PUBO and QUBO problems. For the PUBO formulation, we apply the method presented in Subsubsection 2.3.2 which has the specificity to have binary penalty values. For the QUBO formulation, we apply the second method of Subsection 2.4, which is the most common QUBO reformulation encountered in the literature. For both reformulation, we need to integrate two different types of constraints, (Uni) and (Capa). We recall them below. The first constraint, expressing that a group takes at most one train, is: for all $j \in [m]$,

$$\sum_{\substack{i \in [m]: \\ g_j \in T_i}} y_{ij} \leq 1. \quad (\text{Uni}_j)$$

The second type of constraint, the capacity constraint, is: for all $i \in [n]$,

$$\sum_{\substack{j \in [m]: \\ g_j \in T_i}} y_{ij} \leq \text{CMax} \cdot x_i. \quad (\text{Capa}_i)$$

3.3.1 PUBO reformulation with penalty binary values

Based on Subsection 2.3, and more specifically on the properties of Subsubsection 2.3.2, it results the following penalty terms for the Extended Bin Packing problem as a PUBO.

Proposition 3.2 (Penalty term for Constraint (Uni_j)). *Let $j \in [m]$ and let us note G_j the set of indices of trains accepted by group g_j . The function, for $x \in \{0, 1\}^n$, $y \in \{0, 1\}^q$,*

$$\text{penUni}_j(x, y) = \sum_{k=2}^{|G_j|} (-1)^k (k-1) \sum_{\substack{I=(i_1, \dots, i_k) \subseteq G_j: \\ |I|=k}} y_{i_1, j} \cdots y_{i_k, j}$$

is a penalty term for Constraint (Uni_j).

Proof. Use Property 2.6 for the binary variables y_{ij} such that $i \in G_j$, and for the constant $c = 1$. □

Proposition 3.3 (Penalty term for Constraint (Capa_i)). *Let $i \in [n]$. The following function is a penalty term for Constraint (Capa_i). For $x \in \{0, 1\}^n$, $y \in \{0, 1\}^q$,*

$$\text{penCapa}_i(x, y) = (1 - x_i) \cdot \text{pen}_0(x, y) + x_i \cdot \text{pen}_1(x, y),$$

where

$$pen_0(x, y) = \sum_{k=1}^{|\{j:i \in G_j\}|} (-1)^{k+1} \sum_{\substack{J \subseteq \{j:i \in G_j\}: \\ |J|=k}} y_{i,j_1} \cdots y_{i,j_k}$$

and

$$pen_1(x, y) = \sum_{k=CMax+1}^{|\{j:i \in G_j\}|} (-1)^{k-CMax+1} \binom{k-1}{CMax} \sum_{\substack{J \subseteq \{j:i \in G_j\}: \\ |J|=k}} y_{i,j_1} \cdots y_{i,j_k}.$$

Proof. We distinguish two cases.

- If $x_i = 0$, then we use Property 2.5 for the binary variables y_{ij} such that $j \in \{j : i \in G_j\}$, and for the constant $c = 0$, which provides the term $pen_0(x, y)$.
- If $x_i = 1$, then we use Property 2.6 for the binary variables y_{ij} such that $j \in \{j : i \in G_j\}$, and for the constant $c = CMax$, which provides the term $pen_1(x, y)$.

Eventually, we add the two terms as follows, each one appearing when the condition on x_i is satisfied, to obtain the final penalty function: $(1 - x_i) \cdot pen_0(x, y) + x_i \cdot pen_1(x, y)$. \square

It results from the two propositions above the reformulation of our simplified problem (Extended-BP) into a PUBO problem.

Proposition 3.4 (Extended-BP into PUBO). *The reformulation of (Extended-BP) as a PUBO problem is:*

$$\begin{aligned} \min_{\substack{x \in \{0,1\}^n \\ y \in \{0,1\}^a}} \sum_{i=1}^n c_i x_i - \sum_{i=1}^n \sum_{\substack{j \in [m]: \\ g_j \in T_i}} p_i y_{ij} & \quad (\text{Extended-BP-PUBO}) \\ + \sum_{j=1}^m \lambda_{u,j} penUni_j(x, y) + \sum_{i=1}^n \lambda_{c,i} penCapa_i(x, y), & \end{aligned}$$

where $\lambda_{u,j}, \lambda_{c,i} \in \mathbb{R}_+^*$ for all $j \in [m], i \in [n]$ are the penalty coefficients, and where the penalty functions are defined in Propositions 3.2 and 3.3.

3.3.2 QUBO reformulation

Applying the second method of Subsection 2.4, the penalty terms for the Extended Bin Packing problem as a QUBO are the following.

Proposition 3.5 (Quadratic penalty terms of Constraints (Uni_j) and (Capa_i)). *The penalty term for the first Constraint (Uni_j) is, for $j \in [m]$,*

$$penUniQubo_j(x, y) = \min_{s_j \in \{0,1\}} \left(\sum_{i \in G_j} y_{ij} + s_j - 1 \right)^2.$$

The penalty term for the second constraint (Capa_i) is, for $i \in [n]$,

$$\text{penCapaQubo}_i(x, y) = \min_{r_i \in \llbracket 0, \text{CMax} \rrbracket} \left(\sum_{j: i \in G_j} y_{ij} - \text{CMax} \cdot x_i + r_i \right)^2.$$

In the binary model, the integer slack variables r_i are replaced by their binary expressions $r_i^{\text{bin}} := \sum_{l=0}^{\lfloor \log_2(\text{CMax}) \rfloor} 2^l \cdot r_{i,l}^{\text{bin}}$, for $r_{i,l}^{\text{bin}} \in \{0, 1\}$. In total, the penalty constraints introduce roughly $(m + n \cdot \log_2(\text{CMax}))$ slack binary variables (denoted by s and r) to the initial Extended Bin Packing problem to express the following QUBO formulation.

Proposition 3.6 (Extended-BP into QUBO). *The reformulation of (Extended-BP) as a QUBO problem is*

$$\begin{aligned} \min_{\substack{x \in \{0,1\}^n, \\ y \in \{0,1\}^q, \\ s \in \{0,1\}^m, \\ r^{\text{bin}} \in \{0,1\}^{n \cdot \log_2(\text{CMax})}}} \quad & \sum_{i=1}^n c_i x_i - \sum_{i=1}^n \sum_{\substack{j \in [m]: \\ g_j \in T_i}} p_i y_{ij} & \text{(Extended-BP-QUBO)} \\ & + \sum_{j=1}^m \lambda_{u,j} \left(\sum_{i \in G_j} y_{ij} + s_j - 1 \right)^2 \\ & + \sum_{i=1}^n \lambda_{c,i} \left(\sum_{j: i \in G_j} y_{ij} - \text{CMax} \cdot x_i + r_i^{\text{bin}} \right)^2, \end{aligned} \quad (1)$$

where $\lambda_{u,j}, \lambda_{c,i} \in \mathbb{R}_+^*$ for all $j \in [m], i \in [n]$ are the penalty coefficients.

Not only does the proposed QUBO formulation require additional qubits compared to the PUBO formulation, but it also has penalty terms that take integer values (and not binary values). These two facts are observed and discussed in the next section.

4 Numerical results

In this section, we illustrate the importance of the input problem formulation. Specifically, we compare the results of solving the (Extended-BP-QUBO) and the (Extended-BP-PUBO) reformulations with QAOA on several small instances, showing a trend regarding these two reformulations. For details on QAOA description and implementation, readers can refer to the paper of Grange et al. (2023).

4.1 Instances and results

Let us consider the three instances below (Instance A, Instance B, Instance C). Each set represents a train and each element represents a customer group. In Table 2, we provide for each instance: the numerical values for the problem description (cost of using a train c , benefit of

selling a ticket p , maximum capacity CMax), the optimal solutions and their respective optimal values, and the numerical values of the penalty coefficients (coefficients λ_u, λ_c) for the QUBO and PUBO reformulations. Notice that we only describe the set of trains in the optimal timetable, without specifying the assignment of each customer. The assignment is unique and straightforward for Instances A and B, but for Instance C, there are actually 11 optimal solutions, but still with the 3 sets of trains described in Table 2.

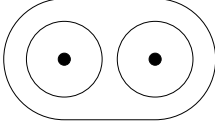


Figure 1: Instance A

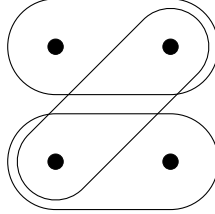


Figure 2: Instance B

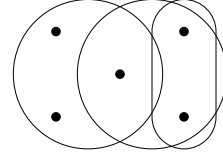


Figure 3: Instance C

	Instance A	Instance B	Instance C
Number of trains	3	3	3
Number of customer groups	2	4	5
Cost $c_i, \forall i \in [n]$	1	1	1
Benefit $p_i, \forall i \in [n]$	1	1	1
Maximum capacity CMax	2	2	2
Optimal solutions (in red)			
Optimal value (f^*)	-1	-2	-2
Range of objective function values ($f_{\min}; f_{\max}$)	$(-4; 3)$	$(-6; 3)$	$(-8; 3)$
QUBO penalty coefficients $\lambda_{u,i} = \lambda_{c,j}, \forall i \in [n], j \in [m]$	8	10	12
PUBO penalty coefficients $\lambda_{u,i} = \lambda_{c,j}, \forall i \in [n], j \in [m]$	8	10	12

Table 2: Description of instances, their optimal solutions and the reformulation coefficients.

In Table 3, we provide the statistics of the two reformulations (QUBO and PUBO) when solving 100 times each of the three instances with QAOA. The details of the results for the 100 runs are displayed in Figure 4. For a run, QAOA returns the best solution encountered during the hybrid optimization, that is the basis state with the smallest loss function value among all the $N_{\text{shots}} \times N_{\text{optim}}$ basis states measured in total, where N_{optim} is the number of iterations of

the classical optimizer, and N_{shots} the number of shots to sample the quantum state at each iteration. Here, N_{shots} is set to 10 to maintain a reasonable ratio between the sample size and the search space size, thus preserving the relevance of our experiments’ scalability. The classical optimizer is chosen to be COBYLA, and no restriction is put on N_{optim} . Finally, the depth of QAOA is set to 1 to ensure that the circuit depth remains manageable relative to the instance size. For each run, we return the loss function value of the unconstrained problem for the solution given by QAOA, and apply the solution to the initial Extended Bin Packing problem to return if it is optimal, feasible non-optimal, or infeasible.

	Instance A		Instance B		Instance C	
	QUBO	PUBO	QUBO	PUBO	QUBO	PUBO
Number of qubits	15	7	17	9	20	11
Proportion of optimal solutions	6%	71%	4%	61%	8%	55%
Proportion of feasible non-optimal solutions	84%	29%	84%	39%	79%	45%
Proportion of infeasible solutions	10%	0%	13%	0%	13%	0%

Table 3: QAOA solution returned for QUBO and PUBO reformulations. The optimality, feasibility non-optimality and infeasibility labels relate to the Extended Bin Packing problem (not yet reformulated).

Notice that we implement QAOA solely using the pre-coded COBYLA optimizer and building the quantum circuits from Qiskit library to be executed on the classical 32-qubit simulator of IBM (backend named *qasm_simulator*). We do not use the pre-coded QAOA function provided by Qiskit and implement, for the PUBO formulation, the quantum circuit with the decomposition proposed by Grange et al. (2023) (Proposition 37). The choice of executing the quantum circuits on a simulator instead on a real quantum hardware is motivated by the fact that today, available quantum backends are still noisy. Thus, to exhibit the impact of the formulation of the input problem on QAOA performances, we decide to use a simulator to *isolate* this parameter from noise and evaluate its real impact.

The PUBO formulation gives better results than the QUBO one on the three instances considered here. Table 3 shows that over 100 runs over each of the 3 instances, QAOA has never returned an unfeasible solution with the PUBO formulation, whereas between 10% and 13% of the solutions returned with the QUBO formulation are infeasible depending on the instances. This means that for some runs of the QUBO formulation, none of the feasible solutions were

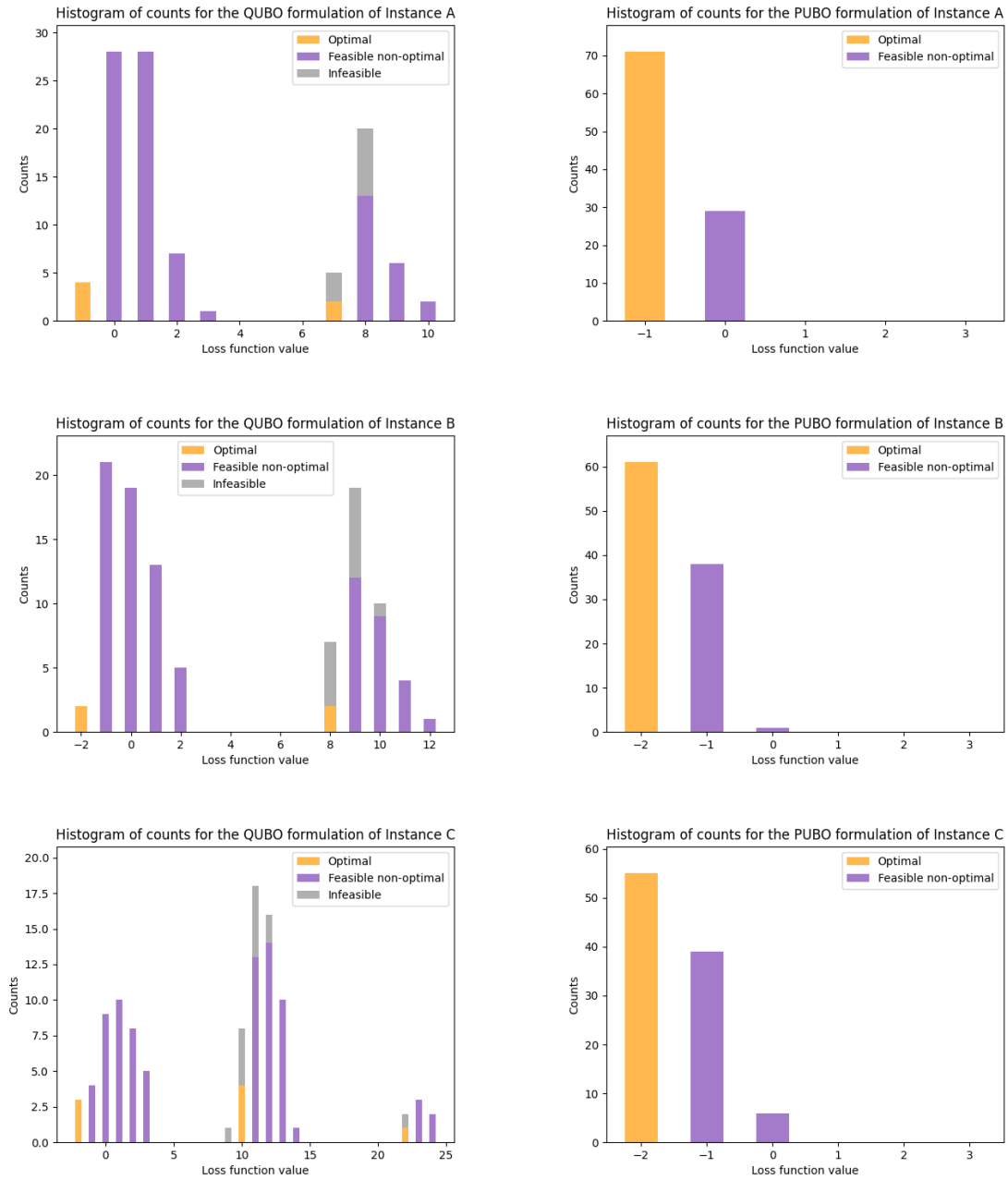


Figure 4: Details of the states returned by running QAOA 100 times for each of the 3 instances (Instance A: row 1, Instance B: row 2, Instance C: row 3) and for each formulation (QUBO: left column, PUBO: right column). The loss function values relate to the unconstrained problem whereas the optimal/feasible non-optimal/infeasible labels' solutions relate to the Extend Bin Packing problem (not yet reformulated).

measured during the whole simulation. Moreover, the PUBO formulation gives far more optimal results than the QUBO formulation: depending on the instances, between 55% and 71% of the runs with the PUBO formulation gave an optimal solution, while it was only 4% to 8% for the

QUBO formulation. Figure 4 provides a first insight into the reasons of those differences. First, notice that the *packets* that can be observed for the QUBO formulation correspond to different values of the penalty functions: each time a penalty function is increased by 1, it adds λ_c or λ_u (depending on the constraint) to the loss function of the unconstrained problem. The PUBO reformulation should present packets as well, but they are not visible in our tests because the PUBO problem is well optimized, and only solutions belonging to the first packet are returned. Those diagrams show that for the PUBO formulation, the optimization always encounters one of the first best solutions in the feasible space, while for the QUBO formulation, the optimizer seems to get lost in the multiple packets of the constraints, and does not always reach the feasible space. Notice that for the QUBO reformulation, optimal solutions from the point of view of the nominal problem do not necessarily minimize the loss function of the unconstrained problem (as mentioned in Subsection 2.4 as *gathering* two minimization problems) because additional variables are not always well optimized. This is why feasible solutions not only appear in the first packet, meaning the one of lower cost. For the PUBO formulation, the packets of higher cost contain only infeasible solutions.

4.2 Discussion

The results of the previous subsection show that the PUBO formulation outperforms the QUBO one for the problem at hand. The possible reasons for this behavior are analyzed below.

Impact of the number of qubits. The first advantage of PUBO reformulations is that they necessitate less qubits than QUBO reformulations. This is a general difference between QUBO and PUBO reformulations, and is not specific to the two particular reformulations which are considered here. Indeed, in the general case, QUBO reformulations use additional variables, thus additional qubits, to integrate the constraints (see Section 2). This difference has two impacts on the quantum part of the hybrid optimization of QAOA. First, PUBO reformulations would be possible on smaller-scale quantum devices. Second, the quantum space to sample is much smaller for PUBO than QUBO as its size evolves exponentially with the number of qubits. Thus, with the same sample size (10 shots here), the probability to measure a good or even optimal solution during the optimization process is enhanced for PUBO reformulations. In our experiment, it is undeniable that the number of qubits plays a role in the advantage of PUBO over QUBO, facilitating the measurement of good solutions at each iteration of the classical optimizer. Our QUBO reformulation uses roughly $(m + n \cdot \log_2(CMax))$ more qubits compared to the PUBO one. For the three instances of our experiment, it correspond approximately to doubling the number of qubits between the PUBO and the QUBO formulations.

Impact of additional variables. As mentioned above, using additional variables for QUBO reformulations has a direct impact on the number of qubits and thus on the size of the quantum circuit. However, it also impacts the classical optimization process. In the nominal constrained optimization problem, the set of constraints defines the feasible space by drawing boundaries in the optimization space. In the PUBO reformulation, the same optimization space is kept, but an important cost is assigned to the loss function when crossing the constraints boundaries. On the other side, the QUBO reformulation increases the dimension of the optimization space with additional variables to assign this cost to the loss function. Consequently, the problem has to be optimized over those new variables as well. When those variables are not correctly optimized, they take the advantage over the cost of the nominal problem in the new loss function because they result in huge penalty terms. Consequently, the relevant information, meaning the value of the nominal loss function, gets *blurred*. This phenomenon can be observed in our experiments: in Figure 4 the optimal solution of the initial problem (in orange) can have a loss function value different from the smallest one possible and even larger than other non-optimal solutions for the QUBO reformulation. This is due to the introduction of the additional variables (s, r^{bin}) in (Extended-BP-QUBO) that can take values such that the penalty terms are non-zero, even if the decision variables (x, y) representing a solution of the initial problem are optimal. Thus, finding an optimal solution for the initial problem embedded in a non-optimal solution of the unconstrained problem is a *coincidence* and has nothing to do with optimization. It points out a limit of the QUBO formulation with additional variables that *blur* the loss function. Notice that this phenomenon does not appear for the PUBO formulation (Extended-BP-PUBO) (on the right column) because there are no additional variables.

Impact of the optimization landscape. As explained before, when a constrained problem is reformulated into an unconstrained one, the uncrossable boundaries of the constraints are replaced by a supplementary cost in the loss function. With the QUBO reformulation, this cost can take a large range of values, and the loss function has to be optimized over it even if it is not meaningful for the problem at stake. As long as the feasible space is not reached, the cost that is optimized has no meaning relatively to the nominal problem. The PUBO problem proposed here is meant to mimic the initial constrained problem, by adding an almost constant step to the optimization landscape when crossing the feasible space boundaries. To achieve this property, the penalty functions are built to take only two values: 0 if the associated constraint is satisfied, 1 otherwise. Depending on the classical optimizer, this simplified landscape could facilitate the optimization. In our experiments, it could be partly responsible for the better results of the PUBO formulation, because COBYLA does not get lost in optimizing over the

infeasible space as for the QUBO formulation.

To summarize, several factors explain the advantage of our PUBO formulation with binary constraints over the generic QUBO formulation. First, the QUBO formulation requires more qubits than the PUBO formulation for the same instance of the nominal problem, leading to a lower probability to measure a good basis state during the optimization. Second, the additional variables necessary for the QUBO reformulation have to be optimized as well. They *blur* the relevant loss function, and add dimensions to the space to be optimized classically. Eventually, these new variables also give rise to a large range of values of the penalty terms, leading to an optimization landscape of the QUBO reformulation which does not reflect the one of the unconstrained initial problem. Those two last points result in an optimization landscape that the classical optimizer does not seem to handle very well according to the three instances solved above. Thus, solving the Extended Bin Packing problem with QAOA achieves better performances with the unconstrained formulation (Extended-BP-PUBO) than with the (Extended-BP-QUBO) formulation on these instances.

5 Conclusion

In this paper, we present generic reformulations of unconstrained combinatorial problems to be solved by Variational Quantum Algorithms, and among them, QAOA. Specifically, we introduce a Polynomial Unconstrained Binary Optimization (PUBO) formulation and two generic Quadratic Unconstrained Binary Optimization (QUBO) formulations, valid for every constrained polynomial optimization problem. In addition, we take advantage of the large possibilities offered by PUBO formulations to propose a formulation of optimization problems with linear constraints into a PUBO problem with binary penalty values. We apply this last formulation and the most common QUBO one on a simplified railway timetabling problem stemming from an industrial problem, and solve each of them with QAOA on several small instances. Our numerical experiments show that the PUBO performs better, drawing a trend for larger instances which haven't been solved yet by lack of high-quality quantum resources. The results speak in favor of the PUBO formulation because it requires less qubits than the QUBO formulation, do not introduce additional variables and seems to present an optimization landscape nicer for the classical optimizer. As soon as large and high-quality quantum hardwares are available, we will be able to deal with the nominal (and more complex) railway timetabling problem of SNCF, comparing PUBO and QUBO formulations on it. This would be the first step of considering quantum algorithms to solve industrial problems.

However, the two reformulations we compare in this paper involve many simultaneous effects, and further studies should be necessary to clearly distinguish their respective impacts. On the one hand, we should study the impact of additional variables by comparing a generic PUBO formulation (Subsubsection 2.3.1) and a generic QUBO formulation (Subsection 2.4). In order to distinguish the impact of additional variables on the classical optimization from the impact of the number of qubits, QAOA could be run with an exact classical computation of the expectation value of the loss function at each iteration of the optimizer. On the other hand, we should analyze the effect of the shape of the penalty functions and the range of values they can take (binary or not) by comparing the two PUBO formulations presented in Subsection 2.3. Future work should also be dedicated to assess the impact of the noise by implementing QAOA on current quantum computers. Working on noisy quantum computers might be advantageous for the QUBO formulation because of a simpler quantum circuit with a smaller depth.

Acknowledgments. This work has been partially financed by the ANRT through the PhD number 2021/0281 with CIFRE funds.

Bibliography

- Ambainis, A., Balodis, K., Iraids, J., Kokainis, M., Prūsis, K., and Vihrovs, J. (2019). Quantum speedups for exponential-time dynamic programming algorithms. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1783–1793. SIAM.
- Azad, U., Behera, B. K., Ahmed, E. A., Panigrahi, P. K., and Farouk, A. (2022). Solving vehicle routing problem using quantum approximate optimization algorithm. *IEEE Transactions on Intelligent Transportation Systems*, 24(7):7564–7573.
- Barkoutsos, P. K., Nannicini, G., Robert, A., Tavernelli, I., and Woerner, S. (2020). Improving variational quantum optimization using CVaR. *Quantum*, 4:256.
- Campbell, C. and Dahl, E. (2022). Qaoa of the highest order. In *2022 IEEE 19th International Conference on Software Architecture Companion (ICSA-C)*, pages 141–146. IEEE.
- Cerezo, M., Arrasmith, A., Babbush, R., Benjamin, S. C., Endo, S., Fujii, K., McClean, J. R., Mitarai, K., Yuan, X., Cincio, L., et al. (2021). Variational quantum algorithms. *Nature Reviews Physics*, 3(9):625–644.
- Dalyac, C., Henriot, L., Jeandel, E., Lechner, W., Perdrix, S., Porcheron, M., and Veshchezerova, M. (2021). Qualifying quantum approaches for hard industrial optimization problems.

- a case study in the field of smart-charging of electric vehicles. *EPJ Quantum Technology*, 8(1):12.
- de la Grand'rive, P. D. and Hullo, J.-F. (2019). Knapsack problem variants of qaoa for battery revenue optimisation. *arXiv preprint arXiv:1908.02210*.
- Farhi, E., Goldstone, J., and Gutmann, S. (2014). A quantum approximate optimization algorithm. *arXiv preprint arXiv:1411.4028*.
- Glover, F., Kochenberger, G., Hennig, R., and Du, Y. (2022). Quantum bridge analytics i: a tutorial on formulating and using qubo models. *Annals of Operations Research*, 314(1):141–183.
- Grange, C., Poss, M., and Bourreau, E. (2023). An introduction to variational quantum algorithms for combinatorial optimization problems. *4OR*, 21(3):363–403.
- Grange, C., Poss, M., Bourreau, E., T'kindt, V., and Ploton, O. (2024). Moderate exponential-time quantum dynamic programming across the subsets for scheduling problems. *European Journal of Operational Research*.
- Hadfield, S., Wang, Z., O'gorman, B., Rieffel, E. G., Venturelli, D., and Biswas, R. (2019). From the quantum approximate optimization algorithm to a quantum alternating operator ansatz. *Algorithms*, 12(2):34.
- Kadowaki, T. and Nishimori, H. (1998). Quantum annealing in the transverse ising model. *Physical Review E*, 58(5):5355.
- Kea, K., Huot, C., and Han, Y. (2023). Leveraging knapsack qaoa approach for optimal electric vehicle charging. *IEEE Access*.
- Kerenidis, I. and Prakash, A. (2020). A quantum interior point method for LPs and SDPs. *ACM Transactions on Quantum Computing*, 1(1):1–32.
- Kurowski, K., Pecyna, T., Slysz, M., Różycki, R., Waligóra, G., and Weglarz, J. (2023). Application of quantum approximate optimization algorithm to job shop scheduling problem. *European Journal of Operational Research*.
- Li, L., Fan, M., Coram, M., Riley, P., Leichenauer, S., et al. (2020). Quantum optimization with a novel Gibbs objective function and ansatz architecture search. *Physical Review Research*, 2(2):023074.
- Lucas, A. (2014). Ising formulations of many NP problems. *Frontiers in Physics*, page 5.

- Montanaro, A. (2020). Quantum speedup of branch-and-bound algorithms. *Physical Review Research*, 2(1):013056.
- Nannicini, G. (2019). Performance of hybrid quantum-classical variational heuristics for combinatorial optimization. *Physical Review E*, 99(1):013304.
- Nannicini, G. (2021). Fast Quantum Subroutines for the Simplex Method. In Singh, M. and Williamson, D. P., editors, *Integer Programming and Combinatorial Optimization - 22nd International Conference, IPCO 2021, Atlanta, GA, USA, May 19-21, 2021, Proceedings*, volume 12707 of *Lecture Notes in Computer Science*, pages 311–325. Springer.
- Preskill, J. (2018). Quantum computing in the NISQ era and beyond. *Quantum*, 2:79.
- Quintero, R., Bernal, D., Terlaky, T., and Zuluaga, L. F. (2022). Characterization of qubo reformulations for the maximum k-colorable subgraph problem. *Quantum Information Processing*, 21(3):89.
- Ruan, Y., Marsh, S., Xue, X., Liu, Z., Wang, J., et al. (2020). The quantum approximate algorithm for solving traveling salesman problem. *Computers, Materials & Continua*, 63(3):1237–1247.
- Salehi, Ö., Glos, A., and Miszczak, J. A. (2022). Unconstrained binary models of the traveling salesman problem variants for quantum optimization. *Quantum Information Processing*, 21(2):67.
- Stein, J., Chamanian, F., Zorn, M., Nüßlein, J., Zielinski, S., Kölle, M., and Linnhoff-Popien, C. (2023). Evidence that pubo outperforms qubo when solving continuous optimization problems with the qaoa. In *Proceedings of the Companion Conference on Genetic and Evolutionary Computation*, pages 2254–2262.
- Tabi, Z., El-Safty, K. H., Kallus, Z., Hága, P., Kozsik, T., Glos, A., and Zimborás, Z. (2020). Quantum optimization for the graph coloring problem with space-efficient embedding. In *2020 IEEE International Conference on Quantum Computing and Engineering (QCE)*, pages 56–62. IEEE.

A Omitted proofs

A.1 Proof of Property 2.5

Property 2.5. *Let us consider the constraint, for $n \in \mathbb{N}$ and $c \in \mathbb{N}$,*

$$\sum_{i=1}^n x_i = c. \quad (\text{Eq})$$

For $c \in \mathbb{N}^$, the penalty function is, for $x \in \{0, 1\}^n$,*

$$\pi_c^{\text{eq}}(x) = 1 + \sum_{k=c}^n (-1)^{k-c+1} \binom{k}{c} \sum_{i=\{i_1, \dots, i_k\} \in I_k^n} x_{i_1} \dots x_{i_k},$$

where I_k^n denotes all the sets of k elements in $[n]$. For the specific case of $c = 0$, the penalty function is

$$\pi_0^{\text{eq}}(x) = \sum_{k=1}^n (-1)^{k+1} \sum_{i \in I_k^n} x_{i_1} \dots x_{i_k}.$$

Proof. Let $x \in \{0, 1\}^n$. Let us begin with the case $c = 0$.

- If x satisfies (Eq), then $\sum_{i=1}^n x_i = 0$ by definition, and it directly results that $\pi_0^{\text{eq}}(x) = 0$.
- If x violates (Eq), then we note $\alpha := \sum_{i=1}^n x_i$. By definition of the violation, $\alpha \in \llbracket 1, n \rrbracket$.

Thus, because for $k > \alpha$, any product of k variables is equal to 0,

$$\begin{aligned} \pi_0^{\text{eq}}(x) &= \sum_{k=1}^{\alpha} (-1)^{k+1} \sum_{i \in I_k^\alpha} x_{i_1} \dots x_{i_k} \\ &= \sum_{k=1}^{\alpha} (-1)^{k+1} \binom{\alpha}{k} \\ &= - \left(\sum_{k=0}^{\alpha} (-1)^k \binom{\alpha}{k} - 1 \right) \\ &= 1 - \sum_{k=0}^{\alpha} (-1)^k 1^{\alpha-k} \binom{\alpha}{k} = 1. \quad (\text{Newton binomial formula}) \end{aligned}$$

Let us next consider the case $c \in \mathbb{N}^*$.

- If x satisfies (Eq), then it exists c variables equal to 1. Let us refer to them as $\hat{x}_1, \dots, \hat{x}_c$. Thus, $\sum_{i \in I_c^n} x_{i_1} \dots x_{i_c} = \hat{x}_1 \dots \hat{x}_c = 1$, and $\sum_{i \in I_k^n} x_{i_1} \dots x_{i_k} = 0$ for $k > c$. It results that

$$\pi_c^{\text{eq}}(x) = 1 + (-1)^{c-c+1} \binom{c}{c} \cdot 1 = 1 - 1 = 0.$$

- If x violates (Eq), then by definition $\sum_{i=1}^n x_i = \alpha \neq c$:
 - If $\alpha < c$, thus $\sum_{i \in I_k^n} x_{i_1} \dots x_{i_k} = 0$ for any $k \geq c$, leading to $\pi_c^{\text{eq}}(x) = 1 - 0 = 1$.

- If $\alpha > c$, thus $\sum_{i \in I_k^n} x_{i_1} \dots x_{i_k} = \binom{\alpha}{k}$ for any $c \leq k \leq \alpha$, and $\sum_{i \in I_k^n} x_{i_1} \dots x_{i_k} = 0$ for any $k > \alpha$. It results that

$$\pi_c^{\text{eq}}(x) = 1 - \sum_{k=c}^{\alpha} (-1)^{k-c+1} \binom{k}{c} \binom{\alpha}{k},$$

where we can show by manipulating factorials that

$$\binom{k}{c} \binom{\alpha}{k} = \binom{\alpha}{c} \binom{\alpha-c}{k-c}.$$

Thus,

$$\begin{aligned} \pi_c^{\text{eq}}(x) &= 1 - \sum_{k=c}^{\alpha} (-1)^{k-c+1} \binom{\alpha}{c} \binom{\alpha-c}{k-c} \\ &= 1 + \binom{\alpha}{c} \sum_{k=0}^{\alpha-c} (-1)^k \binom{\alpha-c}{k} \\ &= 1 + \binom{\alpha}{c} (1-1)^{\alpha-c} = 1. \end{aligned}$$

□

A.2 Proof of Property 2.6

Property 2.6. We consider the constraint, for $n \in \mathbb{N}$ and $c \in \mathbb{N}$,

$$\sum_{i=1}^n x_i \leq c. \quad (\text{Inf})$$

The associated penalty function is, for $x \in \{0, 1\}^n$,

$$\pi_c^{\text{inf}}(x) = \sum_{k=c+1}^n (-1)^{k-c+1} \binom{k-1}{c} \sum_{i \in I_k^n} x_{i_1} \dots x_{i_k}.$$

Proof. Let $x \in \{0, 1\}^n$.

- If x satisfies (Inf), then $\sum_{i \in I_k^n} x_{i_1} \dots x_{i_k} = 0$ for any $k > c$. Thus, $\pi_c^{\text{inf}}(x) = 0$.
- If x violates (Inf), let us note $\sum_{i=1}^n x_i = \alpha > c$. Thus,

$$\sum_{i \in I_k^n} x_{i_1} \dots x_{i_k} = \begin{cases} \binom{\alpha}{k} & \text{for } k \in \llbracket c+1, \alpha \rrbracket \\ 0 & \text{for } k > \alpha \end{cases}$$

It results that

$$\pi_c^{\text{inf}}(x) = \sum_{k=c+1}^{\alpha} (-1)^{k-c+1} \binom{k-1}{c} \binom{\alpha}{k}.$$

Next, we prove by recurrence over $c \in \llbracket 0, \alpha-1 \rrbracket$ the proposition

$$\mathcal{P}(c) : \text{“}\pi_c^{\text{inf}}(x) = 1, \text{ for all } x \in \{0, 1\}^n \text{ violating (Inf)”}.$$

Initialization: $\mathcal{P}(\alpha - 1)$ is True. Indeed, for x violating (Inf),

$$\pi_{\alpha-1}^{\text{inf}}(x) = (-1)^{\alpha-(\alpha-1)+1} \binom{\alpha-1}{\alpha-1} \binom{\alpha}{\alpha} = 1.$$

Recurrence: Let $c \in \llbracket 1, \alpha - 1 \rrbracket$, and let us assume that $\mathcal{P}(c)$ is True. Let us show that $\mathcal{P}(c - 1)$ is also True. For x violating (Inf),

$$\begin{aligned} \pi_{c-1}^{\text{inf}}(x) &= \sum_{k=c}^{\alpha} (-1)^{k-c} \binom{k-1}{c-1} \binom{\alpha}{k} \\ &= (-1)^{c-c} \binom{c-1}{c-1} \binom{\alpha}{c} + \sum_{k=c+1}^{\alpha} (-1)^{k-c} \binom{k-1}{c-1} \binom{\alpha}{k} \\ &= \binom{\alpha}{c} - \sum_{k=c+1}^{\alpha} (-1)^{k-c+1} \left(\binom{k}{c} - \binom{k-1}{c} \right) \binom{\alpha}{k} \quad (\text{Pascal's triangle}) \\ &= \binom{\alpha}{c} - \sum_{k=c+1}^{\alpha} (-1)^{k-c+1} \binom{k}{c} \binom{\alpha}{k} + \underbrace{\sum_{k=c+1}^{\alpha} (-1)^{k-c+1} \binom{k-1}{c} \binom{\alpha}{k}}_{\pi_c^{\text{inf}}(x)} \\ &= \binom{\alpha}{c} - \sum_{k=c+1}^{\alpha} (-1)^{k-c+1} \binom{k}{c} \binom{\alpha}{k} + 1. \quad (\text{Recurrence hypothesis}) \end{aligned}$$

Identically to the proof of the penalty term of constraint (Eq) for the case $c \in \mathbb{N}^*$, we can show that

$$\sum_{k=c+1}^{\alpha} (-1)^{k-c+1} \binom{k}{c} \binom{\alpha}{k} = \binom{\alpha}{c}.$$

Thus, it results that $\pi_{c-1}^{\text{inf}}(x) = \binom{\alpha}{c} - \binom{\alpha}{c} + 1 = 1$.

□

A.3 Proof of Property 2.7

Property 2.7. *We consider the constraint, for $n \in \mathbb{N}$ and $c \in \mathbb{N}^*$,*

$$\sum_{i=1}^n x_i \geq c. \quad (\text{Sup})$$

The associated penalty function is, for $x \in \{0, 1\}^n$,

$$\pi_c^{\text{sup}}(x) = 1 + \sum_{k=c}^n (-1)^{k-c+1} \binom{k-1}{c-1} \sum_{i \in I_k^n} x_{i_1} \dots x_{i_k}.$$

Proof. Let $x \in \{0, 1\}^n$. It is sufficient to show that $\pi_c^{\text{sup}}(x) = \pi_c^{\text{eq}}(x) - \pi_c^{\text{inf}}(x)$, for $c \in \mathbb{N}^*$.

Indeed, if we note $\alpha := \sum_{i=1}^n x_i$, we have the following results.

- If x satisfies (Sup) (meaning that $\alpha \geq c$):
 - If $\alpha = c$: $\pi_c^{\text{eq}}(x) = \pi_c^{\text{inf}}(x) = 0$, thus $\pi_c^{\text{sup}}(x) = 0$.
 - If $\alpha > c$: $\pi_c^{\text{eq}}(x) = \pi_c^{\text{inf}}(x) = 1$, thus $\pi_c^{\text{sup}}(x) = 0$.
- If x violates (Sup) (meaning that $\alpha < c$), we have $\pi_c^{\text{eq}}(x) = 1$ and $\pi_c^{\text{inf}}(x) = 0$, leading to $\pi_c^{\text{sup}}(x) = 1$.

Thus, it remains to prove that $\pi_c^{\text{sup}}(x) = \pi_c^{\text{eq}}(x) - \pi_c^{\text{inf}}(x)$.

$$\begin{aligned}
\pi_c^{\text{eq}}(x) - \pi_c^{\text{inf}}(x) &= 1 + \sum_{k=c}^n (-1)^{k-c+1} \binom{k}{c} \sum_{i \in I_k^n} x_{i_1} \dots x_{i_k} - \sum_{k=c+1}^n (-1)^{k-c+1} \binom{k-1}{c} \sum_{i \in I_k^n} x_{i_1} \dots x_{i_k} \\
&= 1 + (-1)^{c-c+1} \binom{c}{c} \sum_{i \in I_c^n} x_{i_1} \dots x_{i_c} + \sum_{k=c+1}^n (-1)^{k-c+1} \left(\binom{k}{c} - \binom{k-1}{c} \right) \sum_{i \in I_k^n} x_{i_1} \dots x_{i_k} \\
&= 1 - (-1)^{c-c+1} \binom{c-1}{c-1} \sum_{i \in I_c^n} x_{i_1} \dots x_{i_c} + \sum_{k=c+1}^n (-1)^{k-c+1} \binom{k-1}{c-1} \sum_{i \in I_k^n} x_{i_1} \dots x_{i_k} \\
&= 1 + \sum_{k=c}^n (-1)^{k-c+1} \binom{k-1}{c-1} \sum_{i \in I_k^n} x_{i_1} \dots x_{i_k} \\
&= \pi_c^{\text{sup}}(x).
\end{aligned}$$

□