From Constraint to Code: DQI-Kit – A Software Framework for Decoded Quantum Interferometry

Simon Thelen
Technical University of Applied Sciences Regensburg
University of the Bundeswehr Munich
Regensburg/Munich, Germany
simon.thelen@othr.de

Wolfgang Mauerer
Technical University of Applied Sciences Regensburg
Siemens AG, Technology
Regensburg/Munich, Germany
wolfgang.mauerer@othr.de

***Abstract*—Trying to solve hard optimisation problems with quantum techniques requires transformations of domain objectives and constraints into formats compatible with a chosen quantum algorithm. This often introduces inefficiencies and overheads that limit or even endanger potential quantum advantage for current and future approaches. To understand and mitigate these inefficiencies, software toolchains are essential for implementing transformations, analysing overheads and eventually selecting optimal transformation paths. Decoded Quantum Interferometry (DQI) is a novel approach that achieves apparent quantum advantage for certain algebraic optimisation problems. It natively operates on Max-LINSAT, which is unusual for combinatorial optimisation, and creates the need for software solutions that alleviate the burden of manually transforming problems of interest into this format.**

We present DQI-Kit, a software framework that provides a unified, extensible interface for automatically encoding constrained optimisation problems into Max-LINSAT. Users can describe the various types of objectives and constraints that are common in industrial optimisation problems. Our framework converts these into Max-LINSAT instances via a series of problem transformations and computes an estimate of the expected performance of DQI on these instances. We provide an initial analysis of the implemented transformations, discussing inefficiencies and ways to mitigate them. DQI-Kit is the basis for our ultimate goal of establishing a standardised framework that will enable further investigations to identify practical use cases for which quantum advantage with DQI can be achieved.

***Index Terms*—Quantum Software, DQI, Max-LINSAT**

I. Introduction

Owing to their computational difficulty even for small instance sizes, hard combinatorial optimisation problems are considered prime candidates for quantum advantage. Consequently, they are one of the most widely studied aspects of quantum software. Many proposed algorithmic approaches such as quantum annealing and QAOA can be applied to optimisation problems in QUBO format (Quadratic Unconstrained Binary Optimisation) [1]–[6]. This, in principle, allows for approximating solutions to all NP-complete optimisation problems, as the required transformations are well known [7]. Although QUBO formulations are, in practical applications, much less common than classical problem formulations such as Boolean satisfiability (SAT) or mixed-integer linear programs (MILP), extensive research has been conducted on how to bridge the gap between domain problem descriptions and QUBO formulations [8]–[10]. This enables domain experts to apply QUBO-based quantum algorithms to a broad class of industrial use cases.

Decoded Quantum Interferometry (DQI) provides a new algorithmic framework that reduces combinatorial optimisation problems to the decoding problem of a classical error-correcting code [11]. Unlike most previous quantum approaches, which are largely heuristic in nature, the approximation performance of DQI can often be efficiently computed analytically. This allows researchers and engineers to evaluate the potential of DQI in size regimes that would otherwise be inaccessible with current hardware, enabling them to identify problem classes suitable for the algorithm. The optimisation problems native to DQI are Max-XORSAT and its generalisation Max-LINSAT. Formulating domain use cases as Max-LINSAT instances works very differently than for established formulations [12]–[14] like SAT, MILP or QUBO. Consequently, both research and tool support are necessary, as both are currently lacking. Moreover, the solution quality of DQI varies substantially between instances as it depends on properties of an error-correcting code based on the specific Max-LINSAT instance. Thus, assessing the suitability of DQI for a specific problem formulation requires knowledge of coding theory, adding further barriers not only for many quantum researchers, but also for domain users. Published research on potential applications of DQI has mostly focused on coding-theoretic problems that have little to no practical relevance outside of purely algebraic contexts.

Consequently, we introduce *DQI-Kit*, a framework that aims to aid quantum software engineers in overcoming these difficulties. It available in our code repository and reproduction package (links in PDF) [15]. DQI-Kit provides an extensible, unified interface to model various types of constraints and objectives that are common in many industrial combinatorial optimisation problems. Our framework converts such constraints and objectives into DQI-native Max-LINSAT formulations by performing a series of problem transformations, as shown in Figure 1. DQI-Kit also estimates the approximation performance achievable with DQI and compares it to the results of several classical solvers. As visualised by

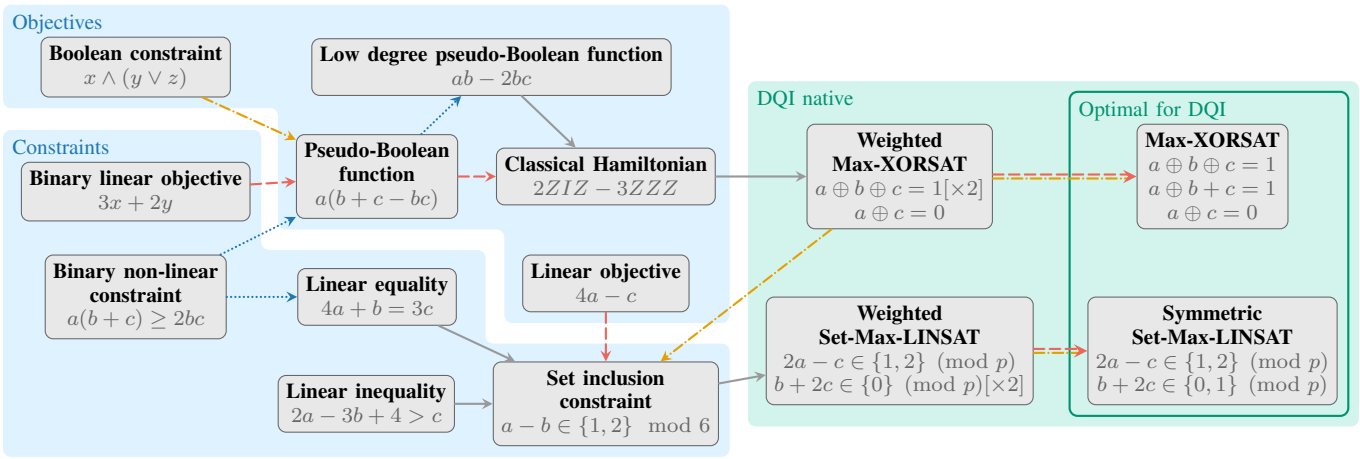


Figure 1: Transformations between formulations implemented by DQI-Kit. Red dashed lines -- represent transformations that increase the number of constraints; blue dotted lines ····· represent transformations that add additional auxiliary variables; yellow dash-dotted lines -·- represent transformations that lead to many linearly dependent constraints often negatively impacting DQI performance; grey solid lines — represent (favourable) transformations where none of the above apply.

the different-coloured arrows in Figure 1, not all problem encodings are equally efficient, highlighting the need for a systematic exploration of the space of possible formulations and transformation paths. We aim to initiate this exploration by providing a first analysis of the efficiency of the encoding approaches implemented by our framework (which is known to be an important issue in general for quantum algorithms [6], [12], [16]–[20]).

The rest of this work is structured as follows: In Section II, we explain the context of our work by presenting the Max-LINSAT problem formulation and by providing brief introductions to both error-correcting codes and the DQI algorithm. Section III covers the problem encodings and transformations implemented by DQI-Kit. In Sections IV and V, we derive guidelines on how to identify problem instances suitable to DQI and options to mitigate some of the limitations of DQI. Section VI presents the DQI-Kit framework. Related work is discussed in Section VII. Finally, we summarise our contributions and give an overview on future work in Section VIII.

II. Preliminaries

A. The Max-LINSAT Problem

The canonical problem that DQI is designed to solve is *Max-LINSAT*, formalised as follows:

Definition 1. *Given a finite field $\mathbb{F}_q$, a matrix $\boldsymbol{B} \in \mathbb{F}_q^{m \times n}$ and a vector $\boldsymbol{v} \in \mathbb{F}_q$,* Max-LINSAT *is the problem of finding a variable assignment $\boldsymbol{x} \in \mathbb{F}_q^n$ that maximises*

$$f(\boldsymbol{x}) = \sum_{i=1}^{m} f_i(\boldsymbol{x}) \text{ with } f_i(\boldsymbol{x}) = \begin{cases} 1 & \text{if } \boldsymbol{x} \cdot \boldsymbol{b}_i = v_i, \\ 0 & \text{otherwise} \end{cases}$$

where $\boldsymbol{b}_i$ denotes the i-th row of $\boldsymbol{B}$ and $\cdot$ denotes the Euclidean inner product.

In other words, a Max-LINSAT instance is defined by a set of m linear constraints over n variables $x_j \in \mathbb{F}_q$, where the i-the constraint is $\sum_{j=1}^{n} B_{ij} x_j = v_i$. The objective is then to find a variable assignment that satisfies as many constraints as possible. Recall that for any finite field $\mathbb{F}_q$, q is either a prime or a prime power. If q is prime, the field $\mathbb{F}_q$ can be defined (up to isomorphism) as the set $\mathbb{Z}_q = \{0, 1, \ldots, q-1\}$ along with addition and multiplication modulo q. Thus, for prime q, a Max-LINSAT instance is a system of linear equations modulo q where the goal is, again, to satisfy as many equations as possible (intuitively, MAX-LinSat can be seen as the optimisation version of solving linear systems over finite fields). The problem is NP-hard in general and independent of q, which makes it an interesting target to study for quantum computing (in cases where all constraints are satisfiable simultaneously, Max-LINSAT can be solved efficiently using Gaussian elimination. So, one typically considers the overdetermined regime for $m > n$).

Due to its similarity to SAT, the special case for $q = 2$ with $x_j \in \{0, 1\}$ has been studied more extensively in the past (*e.g.*, Refs. [21]–[23]). Since addition modulo 2 is equivalent to the binary XOR operation, Max-LINSAT with $q = 2$ is typically referred to as *Max-XORSAT*.

B. Error-Correcting Codes

Let us now briefly review basic definitions and facts from coding theory relevant to determine efficient problem transformations for DQI, as this guides the design of our software architecture. For a more thorough introduction, see, for instance, Refs. [24], [25].

Given some alphabet Σ, an error-correcting code of block length n and dimension k is a subset $\mathcal{C} \subseteq \Sigma^n$ along with an function which injective maps each message $\boldsymbol{m} \in \Sigma^k$ to a codeword $\boldsymbol{c} \in \Sigma^n$ where $n > k$. The goal is to add redundancy by spreading the information of a single message symbol over several codeword symbols. This makes it possible to reconstruct the original codeword and therefore the original

message, even though a few symbols were corrupted during the transmission over a noisy communication channel. Given some corrupted codeword $\tilde{\boldsymbol{c}} \in \Sigma^n$, the goal of a decoder is to find the codeword $\boldsymbol{c} \in \mathcal{C}$ with the smallest Hamming distance to $\tilde{\boldsymbol{c}}$ where the *Hamming distance* is defined as the number of positions at which to symbol strings differ. If the Hamming distance between two distinct codewords $\boldsymbol{c}, \boldsymbol{c}' \in \mathcal{C}$ is small, then a corrupted can be close to both of them, making unique decoding impossible, even with relatively few errors. Thus, one ideally wants the Hamming distance between codewords to be large. In fact, unique decoding of ℓ errors, that is ℓ incorrect symbols, is only possible, in the worst case, if $\ell < d_{\text{min}}/2$. Here, d_{min} denotes the *minimum distance*, which the smallest Hamming distance between distinct codewords in $\mathcal{C}$. Once we try to decode beyond $d_{\text{min}}/2$, there will always be strings for which more than one codeword is within Hamming distance ℓ, making unique decoding impossible. However, for some codes, this situation is rare even for $\ell \geq d_{\text{min}}/2$ as long as ℓ is not too large. For these codes, reliable decoding far beyond half the minimum distance is possible if we can tolerate some small probability of decoding incorrectly.

Linear codes are a special case of error-correcting codes where the alphabet is a finite field $\mathbb{F}_q$ and messages are vectors over this field. The most common field choice is the binary field $\mathbb{F}_2 = \{0, 1\}$. Given a finite field $\mathbb{F}_q$, a linear code $\mathcal{C}$ forms a k-dimensional subspace of the vector space $\mathbb{F}_q^n$, which is defined as the kernel of the so-called *parity-check matrix* $\boldsymbol{H}$. In other words, $\mathcal{C} = \{\boldsymbol{c} \in \mathbb{F}_q^n \mid \boldsymbol{H}\boldsymbol{c} = \boldsymbol{0}\}$. The distance between codewords, and thus the decoding capability of a linear code, is completely determined by its parity-check matrix:

Fact 1. *Given a linear code $\mathcal{C}$ with parity-check matrix $\boldsymbol{H}$ and a codeword $\boldsymbol{c} \in \mathcal{C}$, then there exists a codeword $\boldsymbol{c}' \in \mathcal{C}$ at Hamming distance d of $\boldsymbol{c}$ if and only if $\boldsymbol{H}$ contains a set of d linearly dependent columns. In particular, the minimum distance of $\mathcal{C}$ is equal to the smallest number d_{min} such that $\boldsymbol{H}$ has d_{min} linearly dependent columns.*

This fact is particularly relevant in the context of DQI, as we will see in Section IV, because it allows us to determine a priori how well DQI can approximate the optimal solution of a given Max-LINSAT instance.

The decoding problem for linear codes is NP-complete in general, even in the unique decoding regime. In practice, however, efficient decoding is often possible via specialised decoders that exploit the algebraic structure of specific code families, or via heuristic methods such as belief propagation or information-set decoding. Although these latter approaches do not offer worst-case guarantees, they still deliver excellent decoding performance on a wide range of codes.

C. Decoded Quantum Interferometry

Decoded Quantum Interferometry is a quantum algorithm which approximates solutions to Max-LINSAT instances. It does so by preparing a superposition of all possible solutions that is biased toward good solutions. Thus, when measuring this state, one likely obtains a good solution. Concretely, DQI prepares the *DQI state*:

$$|P(f)\rangle = \sum_{\boldsymbol{x} \in \mathbb{F}_q^n} P(f(\boldsymbol{x}))\,|\boldsymbol{x}\rangle\,. \tag{1}$$

Here, f is the objective function as specified in Definition 1 and P is a polynomial of degree ℓ. The larger the value of ℓ, the stronger the separation between good and bad solutions and the larger the probability of measuring a good solution. However, how large one can choose ℓ to be depends on the error-correcting code $\mathcal{C}$ with parity-check matrix $\boldsymbol{B}^T$ where $\boldsymbol{B}^T$ is the transpose of the constraint matrix $\boldsymbol{B}$ from Definition 1. DQI can only prepare (1) exactly if there is a decoding algorithm for $\mathcal{C}$ that can perfectly correct up to ℓ errors. For imperfect decoders, for example when ℓ is larger than half the minimum distance of $\mathcal{C}$, DQI can still approximate Max-LINSAT solutions by approximating the state (1). However, the solution quality depends on how many error patterns with up to ℓ errors can be decoded correctly.

In the perfect decoding regime and also in the imperfect regime for some instance families, the expected number of satisfied constraints when measuring the DQI state can be efficiently computed classically. This permits analysis of the algorithm on industrial-scale instances using current hardware.

In their study of DQI, Jordan *et al.* [11] consider a generalisation of Max-LINSAT, which we will refer to as *Set-Max-LINSAT*. Here, the $v_i \in \mathbb{F}_q$ are replaced by sets $F_i \subseteq \mathbb{F}_q$. The i-th constraint is satisfied when $\boldsymbol{b}\cdot\boldsymbol{x} \in F_i$. Since each such set inclusion constraint can be decomposed into disjoint equality constraints ($\boldsymbol{b} \cdot \boldsymbol{x} = v$ for all $v \in F_i$), Set-Max-LINSAT describes exactly the same optimisation problems as standard Max-LINSAT. However, combining equality constraints with the same left-hand side, as described by the $\boldsymbol{b}_i$, into a single set inclusion constraint not only reduces the number of constraints but can also improve the approximation performance of DQI for reasons described below in Section IV.

III. Problem Encoding for Max-LINSAT

We now commence by outlining Max-LINSAT formulations for typical objective and constraint types, as they are found in many industrial combinatorial optimisation problems. These formulations are implemented as part of the DQI-Kit software framework. Owing to the multitude of possible encoding approaches proposed in the abundant optimisation literature, we necessarily need to restrict our efforts to the most common input formats. Our extensible software design allows, however, to easily add further encodings, and we anticipate the selection will grow in the future. We discuss some alternative encoding techniques alongside the implemented approaches.

Like QUBOs or other unconstrained problem formulations, Max-LINSAT does not support hard constraints: Every constraint can be broken, and solution quality is measured by the number of broken constraints, which is reasonable for optimisation. However, most industrial optimisation problems include at least some hard constraints that immediately result in invalid solutions when they are broken. To express these in

the Max-LINSAT formalism, we treat them as soft constraints that result in a sufficiently large penalty if broken. This makes it useful to consider a weighted version of Max-LINSAT for encoding constraints:

Definition 2. *Given a finite field $\mathbb{F}_q$, a matrix $\boldsymbol{B} \in \mathbb{F}_q^{m\times n}$, a list of sets $F_1, \dots, F_m$ with $F_i \subseteq \mathbb{F}_q^m$ and a weight vector $\boldsymbol{w} \in \mathbb{Q}^m$ with $w_i > 0$,* Weighted Set-Max-LINSAT *is the problem of finding a vector $\boldsymbol{x} \in \mathbb{F}_q^n$ that maximises*

$$f(\boldsymbol{x}) = \sum_{i=1}^{m} f_i(\boldsymbol{x}) \text{ with } f_i(\boldsymbol{x}) = \begin{cases} w_i & \text{if } \boldsymbol{x}\cdot\boldsymbol{b}_i \in F_i \\ 0 & \text{otherwise.} \end{cases}$$

We discuss ways of dealing with these weights in the context of DQI in Section V. While every weighted Set-Max-LINSAT instance can, in principle, can be converted into an unweighted instance by duplicating constraints, this is usually far from optimal.

A. Equality and Inequality Constraints

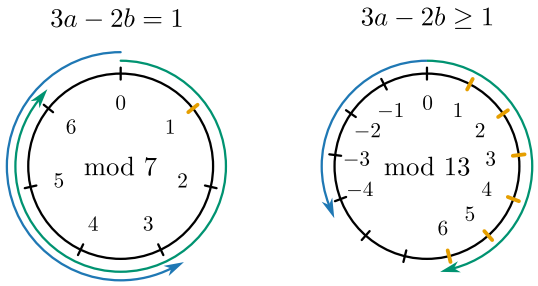
DQI operates over a finite field $\mathbb{F}_q$, meaning each variable has q possible values if not otherwise restricted. This field structure allows for a natural encoding of many types of constraints. For example, one type of constraint directly encodable into Max-LINSAT is an equality constraint $x_i = x_j$, which is expressed as $x_i + (-x_j) = 0$ where $-x_j$ is the additive inverse of x_j in $\mathbb{F}_q$. Beyond that, we can also easily encode inequality constraints of the form $x_i \neq x_j$. These are especially common for categorical variables, for example to describe a scheduling conflict. A 'not equal' constraint is expressed as $x_i + (-x_j) \in \mathbb{F}_q \setminus \{0\}$. This is efficient to encode even for large q since we do not have to represent the set F_i for a given constraint $\boldsymbol{b}_i \cdot \boldsymbol{x} \in F_i$ explicitly during the DQI algorithm. In DQI, each F_i is encoded as is the Fourier transform of a shifted and scaled version of the phase oracle

$$y \mapsto \begin{cases} +1 \text{ if } y \in F_i, \\ -1 \text{ if } y \notin F_i. \end{cases} \tag{2}$$

For simple sets such $F_i = \{0\}$ and $F_i = \mathbb{F}_q \setminus \{0\}$ the number of gates required to implement this phase oracle, and thus the encoding of the F_i during the DQI algorithm, is only logarithmic in q.

Clearly, we can extend these types of equality and inequality constraints to arbitrary linear combinations of variables. If q is prime, can can interpret these as (in)equalities modulo q. However, this interpretation does not hold when $q = p^k$ for a prime p and $k > 1$. This is because the field operations then correspond to addition and multiplication of degree-k polynomials over $\mathbb{F}_p$, which is algebraically distinct from the same operations over the integers modulo q.

Non-modular linear equalities and inequalities over the integers are critical to express many industrial optimisation problems [26], [27]. To encode these, for each integer variable $x_i \in \mathbb{Z}$, we must select a range $l_i, u_i \in \mathbb{Z}$ with $x_i \in [l_i, u_i]$. Then, we select a prime p large enough so all integer constraints can be expressed modulo that prime. Along with the actual constraints we want to encode we also need to add



(a) $=$ or $\neq$ constraint (b) $\leq, <, \geq, >$ constraint

Figure 2: Max-LINSAT encodings of integer constraints: Ordering inequalities generally require a larger field order. In the example, we assume that $a, b \in \{0, 1, 2\}$, so $-4 \leq 3a-2b \leq 6$.

constraints $x_i \in \{l_i, \dots, u_i\} \pmod p$ for all integer variables to restrict their range, each with a sufficiently large weight. Depending on the types of constraints we want to express, p might need to be quite a bit larger then the sizes of the ranges $[l_i, u_i]$. In order for an 'equal' constraint $\boldsymbol{b}_i \cdot \boldsymbol{x} = v_i$ or a 'not equal' constraint $\boldsymbol{b}_i \cdot \boldsymbol{x} \neq v_i$ to also hold modulo p, p must be larger than $\max_{\boldsymbol{x}\in\mathcal{X}} |\boldsymbol{b}_i \cdot \boldsymbol{x} - v_i|$ where $\mathcal{X} = [l_1, u_1] \times \cdots \times [l_m, u_m]$ is the set of allowed values for the $\boldsymbol{x}$. This is visualised in Figure 2a. Ordering equalities ($\leq, <, \geq, >$) are also possible. For example $\boldsymbol{b}_i \cdot \boldsymbol{x} \geq v_i$ can be encoded as the set inclusion constraint $\boldsymbol{b}_i \cdot \boldsymbol{x} - v_i \in \{a \in \mathbb{F}_q \mid a \in \{0, \dots, u_i\} \pmod p\}$. However, for this, p generally has to be larger than for 'equal' or 'not equal' constraints. To be precise, p must be greater than $U_i - L_i$ where $L_i = \min_{\boldsymbol{x}\in\mathcal{X}}(\boldsymbol{b}_i \cdot \boldsymbol{x} - v_i)$ and $U_i = \max_{\boldsymbol{x}\in\mathcal{X}}(\boldsymbol{b}_i \cdot \boldsymbol{x} - v_i)$ as shown in Figure 2b.

If we increase the order of the field $\mathbb{F}_p$ to accommodate non-modular integer constraints, existing modular constraints modulo a smaller prime $p' < p$ need to be modified to conform to the larger field. For example, $\boldsymbol{b}_i \cdot \boldsymbol{x} = v_i \pmod{p'}$ must be replaced with $\boldsymbol{b}_i \cdot \boldsymbol{x} \in \{a \in \mathbb{F}_p \mid a = v_i \pmod{p'}\}$. If $p \neq p'$, then p and p' are coprime. Therefore, p must be greater than $U_i - L_i$ for this to correctly encode the constraint, just like with ordering constraints.

Another option to encode integer constraints is to use a q-ary encoding of integer variables where each Max-LINSAT variable represents a single digit of the integer variable. Using this approach, Sabater *et al.* [28] developed binary-encoded formulations for integer constraints. These are based on Max-XORSAT gadgets implementing a binary ripple-adder. Their approach results in a $\boldsymbol{B}$ matrix with many linearly dependent sets of 3 rows, inhibiting DQI performance, as explained in Section V. For this reason, we chose not to include their encoding technique in the first version of DQI-Kit in favour of our more direct approach, which we deem to be more efficient and broadly applicable.

B. Linear Objectives

The goal of many optimisation problems is to find a variable assignment that maximises or minimises some function under a set of constraints. While Max-LINSAT does not support such objective functions directly, we can still encode them when

allowing weighted constraints. To illustrate this, we can use the constraint $x_i = 1$ for $x_i \in \{0, 1\}$ with weight w_i to represent the objective function $w_i x_i$, which we want to maximise. Similarly, $x_i = 1$ with weight w_i represents the maximising objective $1 - w_i x_i$ or alternatively, ignoring the constant offset of 1, the minimising objective $w_i x_i$. This way, each objective function that is a linear combination of binary variables can be represented by a set of weighted Max-XORSAT constraints.

Encoding linear combinations of non-binary variables is also possible but less efficient: For example, the objective ax_i with $x_i \in \{0, \dots, 6\}$ and $a \in \mathbb{Q}_+$ can be represented by three weighted constraints: $x_i \in \{4, 5, 6\}$ with weight $4a$, $x_i \in \{2, 3, 6\}$ with weight $2a$ and $x_i \in \{1, 3, 5\}$ with weight a. These constraints can be constructed from the binary representation of each of the possible values of x_i, so $\lceil \log_2(u_i - l_i + 1) \rceil$ weighted constraints are necessary for a term ax_i with $x_i \in \{l_i, \dots, r_i\}$.

C. Polynomial Constraints and Objectives

Polynomial constraints and objectives extend linear formulations by allowing interactions between multiple variables, enabling the modelling of higher-order dependencies. Weighted Max-LINSAT can describe maximisation problems of the form $\max_{\boldsymbol{x}} P(\boldsymbol{x})$ where P is some polynomial and $\boldsymbol{x} \in \{0, 1\}^k$ is vector of binary variables. To see this, we can use the fact that Max-XORSAT instances can be interpreted as Hamiltonians that are an unweighted sum of Pauli-Z-Hamiltonians. This equivalence was noticed by [11], [29], [30] among others. In light of Weighted Max-XORSAT, this gives rise to the following generalised equivalence:

Fact 2. *Given a Weighted Max-XORSAT instance as defined in Definition 2, its objective f is equal to $f(\boldsymbol{x}) = \langle \boldsymbol{x} | H | \boldsymbol{x} \rangle$ for the Hamiltonian*

$$H = \frac{W}{2} I^{\otimes n} + \frac{1}{2} \sum_{i=1}^{m} w_i (-1)^{v_i} \prod_{\substack{j=1 \\ B_{ij}=1}}^{m} Z^{(j)}$$

where $W = \sum_{i=1}^{m} w_i$ and $Z^{(i)}$ is the Pauli-Z operator applied to the i-th qubit.

This follows from $f_i(\boldsymbol{x}) = \frac{1}{2}(1 + (-1)^{\boldsymbol{b}_i \cdot \boldsymbol{x} + v_i})$ and the fact that $Z^{(i)} |\boldsymbol{x}\rangle = (-1)^{x_i} |\boldsymbol{x}\rangle$ for all $\boldsymbol{x} \in \{0, 1\}^n$

Since we can ignore the constant offset $W/2$ when trying to find a solution maximising $f(\boldsymbol{x})$, we can, vice versa, convert every Hamiltonian that is diagonal in the Z basis into an equivalent Weighted Max-XORSAT instance. This applies, in particular, to every Ising Hamiltonian and, thus, to every QUBO. Encoding industrial problems as QUBOs is well-established topic in quantum software engineering [1], [7]. The corresponding problem formulations can all, in principle, be directly transformed into Weighted Max-LINSAT instances. By generalising the QUBO reduction to Hamiltonians with Pauli-Z terms of any degree, every polynomial unconstrained binary optimisation (PUBO) problem can be encoded as a Weighted Max-XORSAT instance: Formally, a PUBO is an unconstrained optimisation problem whose objective function $\{0, 1\}^n \to \mathbb{R}$ is given by a multi-variate polynomial of arbitrary degree. In contrast, QUBOs encode polynomials only of degree 2. Every function $b : \{0, 1\}^n \to \mathbb{R}$ can be described as such a multi-variate polynomial [31]. Since $x^2 = x$ for $x \in \{0, 1\}$, this polynomial is always multi-linear, meaning that the exponent of every variable is at most 1 of each of its monomials. Via its representation as a multi-linear polynomial, we can encode every pseudo-Boolean function $b : \boldsymbol{x} \mapsto b(\boldsymbol{x})$ as a Weighted Max-XORSAT instance. To do this, we simply substitute each x_i in the polynomial representation of b with $(1 - Z^{(i)})/2$ and apply the equivalence from Fact 2. However, since a multi-linear polynomial with n variables, in general, consists of 2^n monomials, exponentially many Max-XORSAT constraints are necessary in the worst case. In fact, a simple product $x_1 x_2 \dots x_k$ represented by the Hamiltonian $1/2^k (1 - Z^{(1)}) \dots (1 - Z^{(k)})$ already leads to an exponential number of constraints in k.

There are multiple ways to handle the exponential blow up: For DQI-Kit, we selected a relatively simple technique of introducing additional auxiliary variables that replace high-order sub-terms. We then add constraints to ensure that the auxiliary variables are equal to the terms they represent. To guarantee equality between two expressions $e_1(\boldsymbol{x})$ and $e_2(\boldsymbol{x})$, we add a weighted constraint $-(e_1(\boldsymbol{x}) - e_2(\boldsymbol{x}))^2$, which is maximised when $e_1(\boldsymbol{x}) = e_2(\boldsymbol{x})$. For example, $x_1 x_2 x_3 x_4$ can be split into $x_1 x_2 = a_1$, $x_3 x_4 = a_2$ and $a_1 a_2$ where the two equalities are represented by the terms $-(x_1 x_2 - a_1)^2$ and $-(x_3 x_4 - a_2)^2$. With this approach, the number of Max-XORSAT constraints only grows linearly with the number of variables at the expense of using a potentially linear number of auxiliary variables. In addition to polynomial objectives, this auxiliary variable approach also allows us to express polynomial constraints by first replacing the term with an auxiliary variable and then encoding the constraint, as shown in Section III-A. A more general option to reduce the number of constraints would be to employ a quadratisation algorithm. This similarly reduces monomial degrees at the expense of additional auxiliary variables. Several well-understood quadratisation techniques exist that allow for efficient implementations [18], [32].

Unfortunately, there does not seem to be an obvious way to generalise the encodings of polynomial objectives and constraints explained above to non-binary variables in a way that is compatible with Max-LINSAT. These encodings exploit the group homomorphism $\mathbb{Z}_2 \to \mathbb{Z}, x \mapsto (-1)^x$. The generalisation of this for $p > 2$ is $\mathbb{Z}_p \to \mathbb{C}, x \mapsto \mathrm{e}^{2\pi \mathrm{i} x/p}$. Encoding this into Max-LINSAT is theoretically possible, but would result in complex constraint weights, which are not well-defined. It seems likely that, similarly to MILP formulations, general polynomial constraints and objectives can only be expressed approximately in Max-LINSAT [26], [27].

D. Boolean constraints

The final constraint type we consider is Boolean constraints. Every Boolean constraint, described by a Boolean function $b : \{0, 1\}^n \to \{0, 1\}$, can be transformed into a set of weighted Max-XORSAT constraints by representing b as multi-linear

polynomial and then applying the techniques explained in Section III-C. For example, logical AND is described by the polynomial $(x_1, x_2) \mapsto x_1 x_2$, which is transformed into the Hamiltonian $H = \frac{1}{4}(1 - Z^{(1)})(1 - Z^{(2)})$. This, in turn, is converted into the following Max-LINSAT constraints

$$\begin{aligned} x_1 &= 1 \pmod 2 \\ x_2 &= 1 \pmod 2 \\ x_1 + x_2 &= 0 \pmod 2, \end{aligned} \tag{3}$$

each with weight $1/2$. If both x_1 and x_2 are one, then all 3 constraints are satisfied. Otherwise, only 1 is satisfied. When including the constant offset of $-1/2$, this leads to an objective function, which has value one if $x_1 \wedge x_2$ and zero otherwise. Generalising (3) to n variables leads to the constraints

$$\{\boldsymbol{x} \cdot \boldsymbol{b} = |\boldsymbol{b}| \pmod 2, | \, \boldsymbol{b} \in \{0,1\}^n, |\boldsymbol{b}| > 0\}$$

Here, all $2^n - 1$ constraints are satisfied if $|\boldsymbol{x}| = n$ and $2^{n-1} - 1$ are satisfied otherwise. As can be seen by this example, transforming Boolean functions into Max-XORSAT constraints can and often does lead to the exponential constraint increase described above. The same effect, for example, happens for an n-variable OR constraint, which is transformed into $2^n - 1$ constraints of the form $\boldsymbol{b} \cdot \boldsymbol{x} = 1 \pmod 2$. This necessitates the usage of degree reduction techniques as explained in Section III-C.

IV. Identifying Problems Suited for DQI

Although, as shown in Section III, predominant objective and constraint types can be transformed into Max-LINSAT, not all formulations are equally suited to the DQI algorithm. Finding formulations that work well with DQI requires identifying Max-LINSAT instances where $\mathcal{C} = \{\boldsymbol{y} \mid \boldsymbol{B}^T \boldsymbol{y} = \boldsymbol{0}\}$ is a strong error-correcting code. If we only consider perfect decoders, the relevant quantity for the quality of a code is the minimum distance of $\mathcal{C}$, which, by Fact 1, is the smallest number d such that $\boldsymbol{B}^T$ linearly dependent columns. A set of linearly dependent columns in $\boldsymbol{B}^T$ one-to-one corresponds to a set of linearly dependent rows in $\boldsymbol{B}$. As an illustrative example, consider the following three constraints:

$$\begin{aligned} x_1 + 2x_2 &= 2 \pmod 3 \\ 2x_2 + x_3 &= 1 \pmod 3 \\ x_3 + 2x_1 &= 1 \pmod 3 \end{aligned} \tag{4}$$

The linear combination $\boldsymbol{b}_1 + 2\boldsymbol{b}_2 + \boldsymbol{b}_3$ yields

$$(1,2,0) + 2(0,2,1) + (2,0,1) = (0,0,0) \pmod 3.$$

This means that the linear code corresponding to any Max-LINSAT instance containing these constraints has minimum distance at most 3, independent of the right-hand sides of the constraints. For imperfect decoders, we cannot solely rely on the minimum distance but also need to consider the distance distribution of the code. By Fact 1, we want a code where most codeword pairs have a large distance. This, again, corresponds to a problem matrix $\boldsymbol{B}$ with few linearly dependent rows.

Based on this insight, we can identify specific types of constraints that are likely to impair DQI performance:

1) *Duplicate constraints:* Two constraints $\boldsymbol{b}_i \cdot \boldsymbol{x} = v_i$ and $\boldsymbol{b}_j \cdot \boldsymbol{x} = v_j$ with $\boldsymbol{b}_i = \boldsymbol{b}_j$ immediately lead to a minimum distance of 2, even if $v_i \neq v_j$. This results in a code that, in the worst case, cannot even correct a single error, since we can not distinguish whether an error occurred at position i or j. This severely limits decoders, even in the imperfect decoding regime, likely degrading the approximation performance of DQI significantly.
2) *AND/OR constraints:* Examining the Boolean AND formulation in (3), we observe that the three Max-XORSAT constraints are linearly dependent resulting in a minimum distance of at most 3. The same effect can also be observed for OR constraints.
3) *Short (in)equality cycles:* A cycle of three inequality constraints $x_1 \neq x_2$, $x_2 \neq x_3$, $x_3 \neq x_1$ leads to a situation similar to the outlined in (4), which, in turn, leads to a minimum distance of at most 3. One example of this effect in a practical use case is the Max-cut problem where, given an undirected graph $G = (V, E)$, the objective is to colour each vertex with one of two colours in such a way that the number of edges connecting two vertices of opposite colour is maximised. Max-cut can naturally be encoded as a Max-XORSAT problem by using one variable $x_v \in \mathbb{F}_2$ for each vertex $v \in V$ and adding one constraint $x_u \neq x_v$ for each edge $(u, v) \in E$. It is easy to see that, for this problem formulation, the minimum distance of the code $\mathcal{C}$ is equal to the girth of G, which is defined as the length of the longest cycle. More generally, any cycle

$$l_1 x_1 \sim_1 r_2 x_2, \quad l_2 x_2 \sim_2 r_3 x_3, \quad \ldots, \quad l_k x_k \sim_2 r_1 x_1$$

for non-zero coefficients l_i, r_i and (in)equality relations $\sim_i$ leads to a linear dependency of k constraints.

As these examples show, linear dependencies arise naturally in many practical problem formulations, making it difficult to avoid them entirely. Therefore, in addition to identifying use cases with few inherent dependencies, we deem dependency-reduction techniques to be a promising approach for broadening the applicability of DQI. Below, we demonstrate that mitigating linear dependencies is possible to a certain extent by outlining two concrete techniques. Our aim is for DQI-Kit to facilitate further investigation and refinement of such techniques within the research community.

First, consider the issue of an AND constraint resulting in three linearly dependent Max-LINSAT constraints. This linear dependency can be eliminated by encoding the Boolean constraint only approximately in the sense that the *yes* case corresponds to the optimal variable assignment, while the *no* case corresponds to a sub-optimal but not necessarily worst-possible assignment. For example, if we remove the constraint $x + y = 0 \pmod 2$ removing the linear dependency, the set of constraints is still maximally satisfied (two out of two) when $x \wedge y$. This means that the optimal solution of the Max-LINSAT instance is not affected. As shown by Sabater *et*

al. [28], this approach also works for the three-bit majority gate. It is likely that it can be generalised to various other logical constraints, which could considerably improve DQI's performance on broad classes of problems.

A second, more widely applicable technique involves introducing auxiliary variables to resolve linear dependencies. To illustrate this technique, we present a simple gadget derived from the following result:

Theorem 1. *Let $f(\boldsymbol{x})$ be the objective function of a Max-LINSAT instance with coefficient matrix $\boldsymbol{B} \in \mathbb{F}_q^{m\times n}$ and sets $F_1, \dots, F_m \subseteq \mathbb{F}_q$. Fix an index $i \in \{1, \dots, m\}$ and a positive integer k. Construct a new Max-LINSAT instance by introducing new variables $y_1, \dots, y_k$, replacing the constraint $\boldsymbol{b}_i \cdot \boldsymbol{x} \in F_i$ with $\boldsymbol{b}_i \cdot \boldsymbol{x} + \sum_{j=1}^{k} y_j \in F_i$ and adding constraints $y_j \in \{0\}$ for all $j \in \{1, \dots, k\}$. Let $\hat{f}(\boldsymbol{x}, \boldsymbol{y})$ denote the objective function for this newly constructed instance. Then, for every $\boldsymbol{x} \in \mathbb{F}_q^n$, $k + f(\boldsymbol{x}) = \max_{\boldsymbol{y}\in\mathbb{F}_q^k} \hat{f}(\boldsymbol{x}, \boldsymbol{y})$.*

Proof. For a given $\boldsymbol{x}$, define $v = \boldsymbol{b}_i \cdot \boldsymbol{x}$. If $v \in F_i$, then by setting $y_j = 0$ for $1 \le j \le k$, all $k+1$ new constraints are satisfied, so $k + f(\boldsymbol{x})$ constraints are satisfiable in total.

Otherwise, if $v \notin F_i$, we can either try to satisfy $\boldsymbol{b}_i \cdot \boldsymbol{x} + \sum_j y_j \in F_i$, which requires us to set at least one of the y_j to a non-zero value and satisfying at most k out of the $k+1$ added constraints, or we can satisfy all constraints of the form $y_j = \{0\}$, which leaves $\boldsymbol{b}_i \cdot \boldsymbol{x} + \sum_j y_j \in F_i$ unsatisfied, again satisfying k out of $k+1$ new constraints. In either case, $k + f(\boldsymbol{x})$ constraints are satisfiable in total. □

We can remove linear constraint dependencies by applying Theorem 1: Let $\boldsymbol{b}_1, \dots, \boldsymbol{b}_d$ be a minimal set of linearly dependent rows, in the sense that after removing any single row, the remaining rows are linearly independent. Then, applying the gadget from Theorem 1 to any of the $\boldsymbol{b}_i$ ($1 \le i \le d$) removes the linear dependency but adds a new linear dependency of length $d+k$, which includes the k added constraints $y_j \in \{0\}$. This increases the minimum distance by k at the expense of adding k variables and k constraints. The gadget thus can help improve the approximation performance of DQI by reducing few small linear dependencies. However, introducing many gadgets with large k across the Max-LINSAT instance can quickly lead to diminishing returns. This is because, for large instances, the performance of DQI depends on the ratio ℓ/m [11] and our gadget increases both ℓ and m by k.

In this section, we considered the theoretical decodability of codes derived from Max-LINSAT instances, allowing us to derive universally applicable upper bounds on DQI performance. However, actual performance and runtime of DQI vary depending on the classical decoder, which warrants a more thorough analysis. For instance, belief propagation, the most widely used general purpose decoding algorithm, requires consideration of additional instance properties, such as the sparsity of the constraint matrix and the number of variable pairs shared across constraints [24]. Conversely, specialised decoders tailored to certain problem structures can improve DQI performance considerably [11], [33], [34]. For this reason, DQI-Kit is designed from the outset to support different classical decoders, enabling investigations into the effects of different decoder choices and allowing for a more comprehensive analysis of DQI performance in the future.

V. Handling Limitations of DQI

DQI suffers from two limitations with regards to the problem formulations described in Section III: Firstly, it does not support weighted constraints; secondly, it requires all sets F_i to be of the same size. Both limitations arise because the first step of DQI involves preparing a trial state that is a superposition of all possible error vectors of weight at most ℓ. This trial state is symmetric with respect to different error positions. Due to this symmetry, DQI can only prepare the DQI state (1) exactly if the objective function f is also unbiased with respect to different constraints. This results in symmetry requirements for the problem instance that are only met when all constraints have the same weight and all sets F_i have the same size.

It is plausible that both symmetry requirements can be lifted by using an asymmetric trial state: In a recent work on a generalised version of DQI, Bu *et al.* [35] showed that the DQI state can be prepared efficiently for Weighted Max-XORSAT if $\ell < d_{\min}/2$. This is done by using an asymmetric matrix product state as the trial state, which has bond dimension $\ell+1$ and can thus be prepared efficiently in time polynomial in m and ℓ. This approach may be generalisable for any field $\mathbb{F}_q$ and for sets F_i of different size, but this is not guaranteed. Additionally, computing the expected number of satisfied constraints achieved by DQI is likely less efficient than in the symmetric case, potentially allowing for less precise performance evaluation. Thus, our framework needs to address these symmetry limitations.

The most straightforward (and, in general, only viable) option to handle weighted constraints is to duplicate them such that the number of copies of a constraint is proportional to its weight. However, this creates two problems in turn: First, large weights significantly increases the number of constraints. Second, as explained in Section IV, duplicated constraints heavily limit the approximation capabilities of DQI. To mitigate the first issue, we can divide the weights by a common factor if it is shared by all constraint weights, reducing the number of required duplicates. If the weights do not share a common factor, we can round them to a multiple of an integer d. However, doing so alters the problem we are trying to solve, generally decreasing the solution quality of DQI. Rounding thus works best if weights do not vary a lot between constraints. To mitigate the second problem of duplicate constraints resulting in a minimum distance of 2, we can employ dependency reduction techniques like the the gadget defined in Theorem 1. If we repeat a constraint w_k times, we can resolve the dependency by adding this gadget to $w_k - 1$ of the w_k duplicates.

Finally, we need to handle F_i sets of different sizes. This can also be achieved via duplicate constraints along with dependency reduction techniques. First, we find the greatest common divisor d of all set sizes and split each F_i into sets

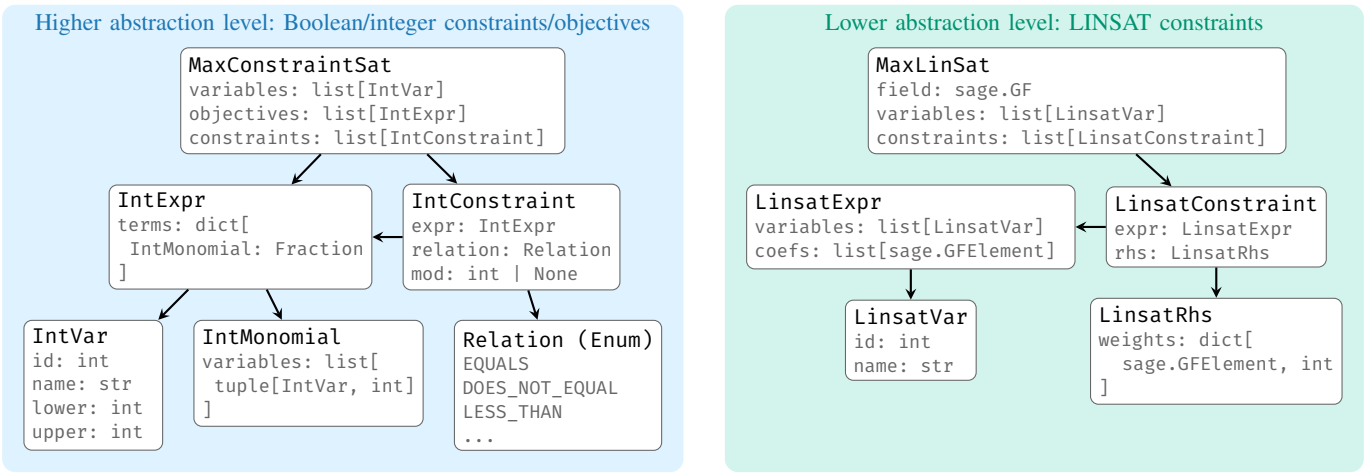


Figure 3: Simplified class structure for the two abstraction levels of DQI-Kit: `MaxLinSat` for directly describing Max-LINSAT constraints and `MaxConstraintSat` for describing integer constraints and objectives as well as Boolean constraints.

of size d. This works especially well for integer constraints as outlined Section III-A. For these types of constraints, we have some flexibility when choosing the sets F_i, since the field order does not usually fit the constraint's range exactly. We can therefore increase the greatest common divisor by deliberately including values in the sets F_i that should not be attainable without violating the variables' range constraints in such a way that the sizes of the F_i have large common factors. In some cases, this approach might still not be viable. It thus remains an interesting open question whether the same-size restriction can be lifted, as was done for the weight restriction in [35].

VI. DQI-Kit

The main contribution of our work is DQI-Kit, an open source Python library to express various types of domain objectives and constraints, which are then converted into weighted or unweighted Max-LINSAT instances via the transformations depicted in Figure 1. DQI-Kit then estimates the approximation performance of both DQI and a series of classical algorithms on these transformed instances.

A. Abstractions for Max-LINSAT Constraints

The general software architecture of our framework is visualised in Figure 3. DQI-Kit provides two main ways for users to describe constraints, which differ in their level of abstraction. Using the lower-level `MaxLinSat` class, users can encode $=$, $\neq$ and $\in$ Max-LINSAT constraints along with their weights directly, allowing for a more manual control of the precise problem formulations. For this, users can choose the finite field $\mathbb{F}_q$ they want to operate in. For the field operations we use *Sage* [36], which provides implementations for arbitrary finite fields via a unified interface to several efficient backends (*e.g.*, Refs. [37]–[39]). Variables are created via `MaxLinSat.new_var`. They always represent elements in $\mathbb{F}_q$. Using Python's operator overloading capabilities, constraints can then be described via intuitive syntax as shown in Figure 4. Equalities and inequalities are rearranged automatically to conform to the Max-LINSAT formalism (Definition 1). For typical constraint patterns beyond $=$, $\neq$ and $\in$ constraints, such as Boolean constraints, `MaxLinSat` supports so-called *gadgets*. A gadget can be thought of as a function mapping a list of n variables to a set of m constraints. It is defined by a miniature Max-LINSAT instance, similar to the one outlined in (3). Users can define their own reusable gadgets. We also provide a small library of gadgets for various Boolean constraints as well as a mechanism to generate a gadget based on a given truth table. This mechanism relies on exhaustively searching all possible sets of constraints. Using it is thus only feasible for small gadgets and field orders.

`MaxLinSat` automatically merges constraints with the same left-hand side into a single constraint. The right-hand side of a constraint is represented as a hash map mapping field elements to weights. For example, merging the constraint $x + y = 0$ with weight 1 and the constraint $x + y \in \{0, 1\}$ with weight 2 results in a constraint that internally represents the right-hand side as the hash map `{0: 3, 1: 2}`. Here, the entry `0: 3` refers to the fact that $x + y$ evaluating to 0 would contribute a total weight of 3 to the Max-LINSAT objective function. Only integer weights are supported. Each `MaxInstance` instance can thus be interpreted as both a weighted and, via constraint duplication, an unweighted Max-LINSAT problem. This allows solvers that do not support weighted constraints, such as standard DQI, to be applied to any `MaxLinSat` instance.

B. Abstractions for Integer and Boolean Constraints

The second abstraction layer provided by DQI-Kit is represented by the class `MaxConstraintSat`. It allows for the specification of various types of (potentially weighted) Boolean, integer and modular constraints. The resulting constraint problem can then be transformed into a `MaxLinSat` instance, via a call to `to_max_lin_sat`. Users can create variables with the `new_var` method. As shown in Figure 3, these variables can be combined with other variables and

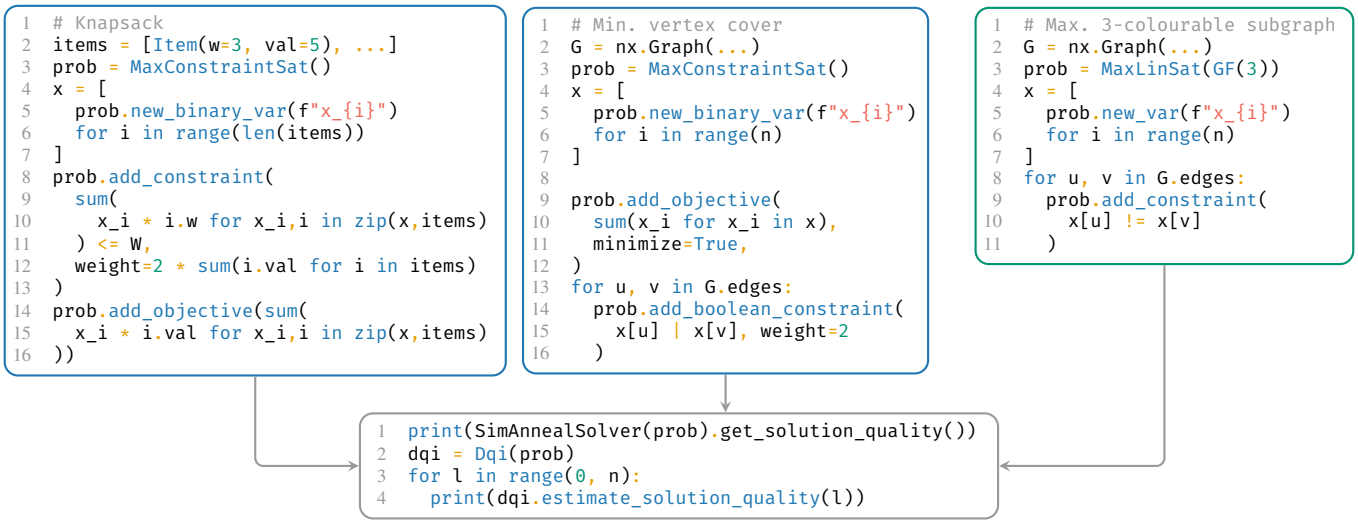


Figure 4: Problem encoding and solution quality estimation in DQI-Kit: User-defined problem specifications supports two abstraction levels: in this example, higher-level `MaxConstraintSat` is used for *knapsack* and *minimum vertex cover*; lower-level abstraction `MaxLinSat` is used for *maximum 3-colourable subgraph*. DQI and classical solvers operate on both. For this, `MaxConstraintSat` instances are automatically transformed into Max-LINSAT.

scalar constants, represented by the Fraction class, to form expressions (`IntExpr`) via addition and multiplication. The set of representable expressions is therefore equal to the set of multi-variate polynomials. An `IntExpr` can be used directly as objective to be maximised or minimised, or, along with a relation $(=, \neq, \leq, <, \geq, >)$, as part of an `IntConstraint`. When describing a modular constraint, only $=$ and $\neq$ are supported since there is no well-defined notion of order in $\mathbb{Z}_n$. By convention, the right-hand side of any `IntConstraint` is 0 and expressions are automatically rearranged accordingly.

`MaxConstraintSat` has no explicit notion of a binary variable. At creation, the value range of each `IntVar` is specified via integer lower and upper bounds. A variable is considered binary if its lower bound is 0 and its upper bound is 1. In addition to `+`, `-` and `*`, binary variables also support the Python Boolean operators `~`, `&`, `|` and `^`, representing NOT, AND, OR and XOR respectively. These Boolean operators are automatically converted to addition and multiplication. For example, $a \vee b$ is equivalent to $a + b - ab$. This means that, internally, a Boolean constraint is handled identically to an integer expressions and is therefore added as an objective and not as a constraint since it includes no relation such as $=$.

The transformation of `MaxConstraintSat` to `MaxLinSat` happens in multiple passes. In the first pass, polynomial objectives and constraints as well as linear objectives are transformed into linear constraints, as outlined in Sections III-B and III-C. As explained in Section III-C, a pseudo-Boolean function over k variables can lead to up to 2^k Max-LINSAT constraints. To combat this, the transformation splits up higher order terms introducing auxiliary variables. This reduces monomial degrees, and thus the number of Max-LINSAT constraints required. By default, terms of degree at least four are split up, but this behaviour can be changed by the user. Auxiliary variables are also introduced to represent higher order terms within constraints.

After the first pass, only linear constraints remain. These are transformed into modular constraints. If needed, constraints are scaled such that they only contain integer coefficients. Then, the smallest prime p is determined so all constraints can be encoded modulo that prime and the integer constraints are converted into constraints over $\mathbb{F}_p$. Next, for each variable x_i with lower bound l_i and upper bound u_i with $u_i - l_i + 1 < p$, a constraint $x_i \in \{l_i, \ldots, u_i\}$ is added. Finally, we end up with three types of constraints: user-defined constraints, auxiliary variable constraints for higher-order terms and variable range constraints. The latter two types are automatically assigned weights that are large enough to give them priority over user-defined constraints, unless the user specifies otherwise.

C. Solvers and Decoders

Our software framework implements a DQI solver and four classical solvers for Max-LINSAT instances. Actually executing DQI on a real or simulated quantum computer is impossible with current hardware, for industrial-scale instances. Instead of performing the algorithm, the DQI solver therefore computes the expected solution quality in terms of the expected number of satisfied constraints using the closed expressions derived by Jordan *et al.* [11]. The user can choose (a) the degree ℓ of the polynomial P (which determines how strongly the final superposition prepared by DQI is biased towards the optimal solution), and (b) the classical decoder used during syndrome decoding. We provide four general-purpose decoding algorithms (belief propagation, information set decoding, lookup table syndrome decoding and nearest-neighbour decoding) as well as an interface to add other decoders. We use the `ldpc` package [40] for belief

propagation and Sage [36] for the other decoders. Note that the lookup table syndrome decoder and the nearest-neighbour decoder correct all theoretically correctable errors but they have exponential runtime in ℓ. Nevertheless, they can efficiently provide provable upper bounds on problem instances with small minimum distance making them useful for ruling out potential problem candidates for DQI. If the user does not specify ℓ or the decoder, both are selected automatically depending on the order of the field $\mathbb{F}_q$ and the approximate minimum distance of the linear code $\mathcal{C}$ dual to the given Max-LINSAT instance. If not otherwise specified, DQI-Kit also selects the optimal coefficients of P automatically by solving an eigenvalue problem as described in [11].

If ℓ is smaller than half the minimum distance of $\mathcal{C}$, we can efficiently compute the expected number of satisfied constraints exactly. Otherwise, computing this quantity either takes exponential time or can only be done approximately by sampling from the distribution of possible error vectors. In DQI-Kit, the user can select if they want an exact or approximated value and, in the latter case, how many errors should be sampled. This give a more accurate approximation at the expense of compute time.

Along with the DQI solver, we provide interfaces for four classical methods: a brute force solver for finding the optimal solution, the constraint programming solver from Googles OR-Tools [41], a simulated annealing solver using the `simanneal` package [42] and, finally, Prange's algorithm. Prange's algorithm finds an approximate solution by randomly selecting a set of n constraints, whose left-hand sides are linearly independent. The resulting system of equations is then solved exactly using Gaussian elimination. If we assume the $m-n$ remaining constraints to be random and independent from the n selected, this satisfies $n+(m-n)/q$ equations in total, on average. Prange's algorithm is noteworthy as it is the currently best known classical algorithm for the problem for which [11] showed apparent quantum advantage with DQI.

VII. Related Work

DQI was proposed by Jordan *et al.* [11] as a quantum algorithm that finds approximate solutions to a combinatorial optimisation problem via so-called *Regev reductions*. This type of reduction is based on the relationship via a Fourier transform between decoding problems in two codes dual to each other. Regev reductions have has been studied before DQI, both framed in the context of lattice problems [43]–[45] and coding theory [46]–[48]. There have been several works building upon the original DQI paper [29], [33], [34], [49], [50]. In particular, we want to highlight the works of Anschuetz *et al.* [51] and Kramer *et al.* [52], who provide compelling evidence indicating that quantum advantage with DQI is not achievable on unstructured instances. We also want to highlight the work of Parekh [30], who ruled out quantum advantage for Max-Cut with DQI when using perfect decoding algorithms. These results underscore the importance of identifying suitable problem instances for utilising DQI in industrial applications. DQI has been successfully applied to algebraic problems derived from known good error-correcting codes [11], [33], [34], [49], [53]. However, to the best of our knowledge, Sabater *et al.* [28] published the first investigation of DQI on an industrial use case. They apply DQI to a binary integer linear program and but find that, for their problem formulation, DQI is not competitive against state-of-the-art classical approaches.

Software-aided transformations of domain problems [54], [55] into mathematical formalisms have been broadly studied for other problem formulations: For instance, interfaces to encode various types of constraints are provided by many state-of-the-art mixed-integer linear programming solvers including Gurobi, Google OR-Tools or the SCIP Optimisation Suite [41], [56], [57]. In addition, there has been extensive research on the development of domain-specific languages for problem encoding [58]–[60]. In quantum optimisation, QUBOs the most widely studied problem formulation. Multiple efforts provide tools to encode various constraints as QUBOs, which, in turn, can then be used in quantum approaches such as QAOA or quantum annealing [8], [9], [61]–[63].

VIII. Conclusion and Outlook

We presented DQI-Kit, an open source software framework that allows users to study the performance of the DQI algorithm on broad classes of industrial combinatorial optimisation problems. We demonstrated that high-level user-specified descriptions of predominant constraint and objective types can be encoded into problem formulations directly compatible with DQI via a series of transformations. Based on the limitations of DQI, we derived a series of guidelines on how to determine suitable problem instances and on which types of constraints can degrade the performance of the algorithm.

Our work shows that many transformations introduce inefficiencies in the form of auxiliary variables, an increased number of constraints or linear dependencies in the constraint matrix. These inefficiencies can limit the performance of DQI, sometimes significantly. We demonstrate that, in principle, mitigating some of these inefficiencies is possible to a certain extent. To significantly expand the class of problems where DQI is applicable, further research into more efficient transformations and more sophisticated dependency mitigation techniques is required. Developing generalisations of DQI for weighted constraints or heterogeneous right-hand side sets could also extend its applicability considerably by allowing for more efficient encoding of many practical use cases. We envision DQI-Kit as a starting point for a community-driven standard that provides a unified interface for a wide range of possible transformation and encoding strategies.

Acknowledgements This work was supported by the German Federal Ministry of Research, Technology and Space (BMFTR), funding program 'Research Program Quantum Systems', grant number 13N16092. We acknowledge partial support by the German Research Foundation, grant MA 9739/1-1, by the European Union (Project Reference 101083427) and the European Funds for Regional Development (EFRE) (Project Reference 20-3092.10-THD-105). WM acknowledges support by the High-Tech Agenda Bavaria.

References

[1] S. Yarkoni, E. Raponi, T. Bäck, and S. Schmitt, "Quantum annealing for industry applications: Introduction and review," *Reports on Progress in Physics*, vol. 85, no. 10, p. 104001, 2022.

[2] K. Blekos, D. Brand, A. Ceschini, C.-H. Chou, R.-H. Li, K. Pandya, and A. Summer, "A review on quantum approximate optimization algorithm and its variants," *Physics Reports*, vol. 1068, p. 1–66, Jun. 2024. [Online]. Available: http://dx.doi.org/10.1016/j.physrep.2024.03.002

[3] S. Thelen and W. Mauerer, "Predict and conquer: Navigating algorithm trade-offs with quantum design automation," in *IEEE International Conference on Quantum Computing and Engineering (QCE)*, vol. 01, 2025, pp. 591–602.

[4] S. Thelen, H. Safi, and W. Mauerer, "Approximating under the influence of quantum noise and compute power," in *IEEE International Conference on Quantum Computing and Engineering (QCE)*, vol. 02, 2024, pp. 274–279.

[5] A. and Bayerstadler, G. Becquin, J. Binder, T. Botter, H. Ehm, T. Ehmer, M. Erdmann, N. Gaus, P. Harbach, M. Hess, J. Klepsch, M. Leib, S. Luber, A. Luckow, M. Mansky, W. Mauerer, F. Neukart, C. Niedermeier, L. Palackal, R. Pfeiffer, C. Polenz, J. Sepulveda, T. Sievers, B. Standen, M. Streif, T. Strohm, C. Utschig-Utschig, D. Volz, H. Weiss, and F. Winter, "Industry quantum computing applications," *EPJ Quantum Technology*, vol. 8, no. 1, Nov. 2021. [Online]. Available: http://dx.doi.org/10.1140/epjqt/s40507-021-00114-x

[6] K. Bharti, A. Cervera-Lierta, T. H. Kyaw, T. Haug, S. Alperin-Lea, A. Anand, M. Degroote, H. Heimonen, J. S. Kottmann, T. Menke, W.-K. Mok, S. Sim, L.-C. Kwek, and A. Aspuru-Guzik, "Noisy intermediate-scale quantum algorithms," *Rev. Mod. Phys.*, vol. 94, p. 015004, Feb 2022. [Online]. Available: https://link.aps.org/doi/10.1103/RevModPhys.94.015004

[7] A. Lucas, "Ising formulations of many np problems," *Frontiers in Physics*, vol. 2, 2014. [Online]. Available: http://dx.doi.org/10.3389/fphy.2014.00005

[8] M. Zaman, K. Tanahashi, and S. Tanaka, "Pyqubo: Python library for qubo creation," *IEEE Transactions on Computers*, 2021.

[9] E. Lobe, "quark: Quantum application reformulation kernel," 2023. [Online]. Available: https://dl.gi.de/handle/20.500.12116/43045

[10] N. Franco, T. Wollschläger, B. Poggel, S. Günnemann, and J. M. Lorenz, "Efficient milp decomposition in quantum computing for relu network robustness," in *2023 IEEE International Conference on Quantum Computing and Engineering (QCE)*, vol. 01, 2023, pp. 524–534.

[11] S. P. Jordan, N. Shutty, M. Wootters, A. Zalcman, A. Schmidhuber, R. King, S. V. Isakov, T. Khattar, and R. Babbush, "Optimization by decoded quantum interferometry," *Nature*, vol. 646, no. 8086, p. 831–836, Oct. 2025. [Online]. Available: http://dx.doi.org/10.1038/s41586-025-09527-5

[12] L. Schmidbauer and W. Mauerer, "Sat strikes back: Parameter and path relations in quantum toolchains," in *2025 IEEE International Conference on Quantum Software (QSW)*. IEEE, Jul. 2025, p. 01–12. [Online]. Available: http://dx.doi.org/10.1109/QSW67625.2025.00021

[13] T. Gabor, S. Zielinski, S. Feld, C. Roch, C. Seidel, F. Neukart, I. Galter, W. Mauerer, and C. Linnhoff-Popien, "Assessing solution quality of 3sat on a quantum annealing platform," in *Quantum Technology and Optimization Problems*, S. Feld and C. Linnhoff-Popien, Eds. Cham: Springer International Publishing, 2019, pp. 23–35. [Online]. Available: https://arxiv.org/abs/1902.04703

[14] T. Krüger and W. Mauerer, "Quantum annealing-based software components: An experimental case study with sat solving," in *Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops*, ser. ICSEW'20. New York, NY, USA: Association for Computing Machinery, 2020, p. 445–450. [Online]. Available: https://doi.org/10.1145/3387940.3391472

[15] W. Mauerer and S. Scherzinger, "1-2-3 reproducibility for quantum software experiments," in *IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2022, pp. 1247–1248.

[16] M. Gogeissl, H. Safi, and W. Mauerer, "Quantum data encoding patterns and their consequences," in *Proceedings of the 1st Workshop on Quantum Computing and Quantum-Inspired Technology for Data-Intensive Systems and Applications*, ser. Q-Data '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 27–37. [Online]. Available: https://doi.org/10.1145/3665225.3665446

[17] M. Schönberger, I. Trummer, and W. Mauerer, "Quantum-inspired digital annealing for join ordering," in *Proceedings of the VLDB Endowment*, vol. 17, no. 3, 11 2023. [Online]. Available: https://doi.org/10.14778/3632093.3632112

[18] L. Schmidbauer, E. Lobe, I. Schaefer, and W. Mauerer, "It's quick to be square: Fast quadratisation for quantum toolchains," 02 2026. [Online]. Available: https://arxiv.org/abs/2411.19934

[19] L. Schmidbauer, K. Wintersperger, E. Lobe, and W. Mauerer, "Polynomial reduction methods and their impact on qaoa circuits," in *2024 IEEE International Conference on Quantum Software (QSW)*, 2024, pp. 35–45.

[20] L. Schmidbauer, C. A. Riofrío, F. Heinrich, V. Junk, U. Schwenk, T. Husslein, and W. Mauerer, "Path matters: Industrial data meet quantum optimization," in *Proceedings of the IEEE International Conference on Quantum Computing and Engineering*, 5 2025. [Online]. Available: https://arxiv.org/abs/2504.16607

[21] M. Mézard, F. Ricci-Tersenghi, and R. Zecchina, "Two solutions to diluted p-spin models and xorsat problems," *Journal of Statistical Physics*, vol. 111, no. 3, pp. 505–533, 2003.

[22] M. Ibrahimi, Y. Kanoria, M. Kraning, and A. Montanari, "The set of solutions of random xorsat formulae," in *Proceedings of the twenty-third annual ACM-SIAM symposium on Discrete Algorithms*. SIAM, 2012, pp. 760–779.

[23] B. Pittel and G. B. Sorkin, "The satisfiability threshold for k-xorsat," *Combinatorics, Probability and Computing*, vol. 25, no. 2, pp. 236–268, 2016.

[24] S. Lin and J. Li, *Fundamentals of Classical and Modern Error-Correcting Codes*. Cambridge University Press, 2021.

[25] V. Guruswami, A. Rudra, and M. Sudan, *Essential coding theory*, 2025.

[26] G. Nemhauser and L. Wolsey, *Integer and Combinatorial Optimization*. Wiley, Jun. 1988. [Online]. Available: http://dx.doi.org/10.1002/9781118627372

[27] H. P. Williams, *Model building in mathematical programming*, 5th ed. Hoboken, NJ: Wiley-Blackwell, Feb. 2013.

[28] F. Sabater, O. E. Harzli, G.-J. Besjes, M. Erdmann, J. Klepsch, J. Hiltrop, J.-F. Bobier, Y. Cao, and C. A. Riofrio, "Towards solving industrial integer linear programs with decoded quantum interferometry," 2025. [Online]. Available: https://arxiv.org/abs/2509.08328

[29] A. Schmidhuber, J. Z. Lu, N. Shutty, S. Jordan, A. Poremba, and Y. Quek, "Hamiltonian decoded quantum interferometry," 2025. [Online]. Available: https://arxiv.org/abs/2510.07913

[30] O. Parekh, "No quantum advantage in decoded quantum interferometry for maxcut," 2025. [Online]. Available: https://arxiv.org/abs/2509.19966

[31] P. L. Hammer and S. Rudeanu, *Boolean Methods in Operations Research and Related Areas*. Springer Berlin Heidelberg, 1968. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-85823-9

[32] N. Dattani, "Quadratization in discrete optimization and quantum mechanics," 2019. [Online]. Available: https://arxiv.org/abs/1901.04405

[33] A. Gu and S. P. Jordan, "Algebraic geometry codes and decoded quantum interferometry," 2025. [Online]. Available: https://arxiv.org/abs/2510.06603

[34] T. Khattar, N. Shutty, C. Gidney, A. Zalcman, N. Yosri, D. Maslov, R. Babbush, and S. P. Jordan, "Verifiable quantum advantage via optimized dqi circuits," 2025. [Online]. Available: https://arxiv.org/abs/2510.10967

[35] K. Bu, W. Gu, and X. Li, "Hamiltonian decoded quantum interferometry for general pauli hamiltonians," 2026. [Online]. Available: https://arxiv.org/abs/2601.18773

[36] The Sage Developers, *SageMath, the Sage Mathematics Software System (Version 10.8)*, 2025. [Online]. Available: https://www.sagemath.org

[37] *PARI/GP version – version 2.15.4*, The PARI Group, Univ. Bordeaux, 2023. [Online]. Available: http://pari.math.u-bordeaux.fr/

[38] V. Shoup, "Ntl – a library for doing numbery theory – version 11.6.0," 2025. [Online]. Available: https://github.com/libntl/ntl/releases/tag/v11.6.0

[39] *Givaro: C++ library for arithmetic and algebraic computations – Version 4.2.1*, The Givaro Group, 2025. [Online]. Available: https://github.com/linbox-team/givaro/releases/tag/v4.2.1

[40] J. Roffe. (2022) LDPC: Python tools for low density parity check codes. [Online]. Available: https://pypi.org/project/ldpc/

[41] L. Perron and F. Didier. Cp-sat. Google. [Online]. Available: https://developers.google.com/optimization/cp/cp_solver/

[42] M. Perry and R. J. Wagner. simanneal. [Online]. Available: https://pypi.org/project/simanneal/

[43] D. Aharonov and A. Ta-Shma, "Adiabatic quantum state generation and statistical zero knowledge," in *Proceedings of the Thirty-Fifth Annual ACM Symposium on Theory of Computing*, ser. STOC '03. New York, NY, USA: Association for Computing Machinery, 2003, p. 20–29. [Online]. Available: https://doi.org/10.1145/780542.780546

[44] D. Aharonov and O. Regev, "Lattice problems in np ∩ conp," *J. ACM*, vol. 52, no. 5, p. 749–765, Sep. 2005. [Online]. Available: https://doi.org/10.1145/1089023.1089025

[45] O. Regev, "On lattices, learning with errors, random linear codes, and cryptography," *J. ACM*, vol. 56, no. 6, Sep. 2009. [Online]. Available: https://doi.org/10.1145/1568318.1568324

[46] Y. Chen, Q. Liu, and M. Zhandry, "Quantum algorithms for average-case lattice problems via filtering," in *Advances in Cryptology – EUROCRYPT 2022: 41st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Trondheim, Norway, May 30 – June 3, 2022, Proceedings, Part III*. Berlin, Heidelberg: Springer-Verlag, 2022, p. 372–401. [Online]. Available: https://doi.org/10.1007/978-3-031-07082-2_14

[47] T. Yamakawa and M. Zhandry, "Verifiable quantum advantage without structure," *J. ACM*, vol. 71, no. 3, Jun. 2024. [Online]. Available: https://doi.org/10.1145/3658665

[48] T. Debris-Alazard, M. Remaud, and J.-P. Tillich, "Quantum reduction of finding short code vectors to the decoding problem," *IEEE Trans. Inf. Theor.*, vol. 70, no. 7, p. 5323–5342, Jul. 2024. [Online]. Available: https://doi.org/10.1109/TIT.2023.3327759

[49] A. Chailloux and J.-P. Tillich, "Quantum advantage from soft decoders," in *Proceedings of the 57th Annual ACM Symposium on Theory of Computing*, ser. STOC '25. New York, NY, USA: Association for Computing Machinery, 2025, p. 738–749. [Online]. Available: https://doi.org/10.1145/3717823.3718319

[50] K. Marwaha, B. Fefferman, A. Gheorghiu, and V. Havlicek, "On the complexity of decoded quantum interferometry," 2025. [Online]. Available: https://arxiv.org/abs/2509.14443

[51] E. R. Anschuetz, D. Gamarnik, and J. Z. Lu, "Decoded quantum interferometry requires structure," 2025. [Online]. Available: https://arxiv.org/abs/2509.14509

[52] M. J. Kramer, C. Schubert, and J. Eisert, "Tight inapproximability of max-linsat and implications for decoded quantum interferometry," 2026. [Online]. Available: https://arxiv.org/abs/2603.04540

[53] D. C. Hillel, "Optimization of quadratic constraints by decoded quantum interferometry," 2025. [Online]. Available: https://arxiv.org/abs/2510.08061

[54] T. Yue, W. Mauerer, S. Ali, and D. Taibi, *Challenges and Opportunities in Quantum Software Architecture*. Springer Nature Switzerland, 2023, p. 1–23. [Online]. Available: http://dx.doi.org/10.1007/978-3-031-36847-9_1

[55] C. Carbonelli, M. Felderer, M. Jung, E. Lobe, M. Lochau, S. Luber, W. Mauerer, R. Ramler, I. Schaefer, and C. Schroth, "Challenges for Quantum Software Engineering: An Industrial Application Scenario Perspective," in *Quantum Software*, I. Exman, R. Pérez-Castillo, M. Piattini, and M. Felderer, Eds. Cham: Springer Nature Switzerland, 2024, pp. 311–335. [Online]. Available: https://doi.org/10.1007/978-3-031-64136-7_12

[56] Gurobi Optimization, LLC, "Gurobi Optimizer Reference Manual," 2026. [Online]. Available: https://www.gurobi.com

[57] C. Hojny, M. Besançon, K. Bestuzheva, S. Borst, J. Dionísio, J. Ehls, L. Eifler, M. Ghannam, A. Gleixner, A. Göß, A. Hoen, J. von Holly-Ponientzietz, R. van der Hulst, D. Kamp, T. Koch, K. Kofler, J. Lentz, M. Lübbecke, S. J. Maher, P. M. Meinhold, G. Mexi, T. Mohr, E. Mühmer, K. K. Patel, M. E. Pfetsch, S. Pokutta, C. R. Groba, F. Serrano, Y. Shinano, M. Turner, S. Vigerske, M. Walter, D. Weninger, and L. Xu, "The scip optimization suite 10.0," 2025. [Online]. Available: https://arxiv.org/abs/2511.18580

[58] N. Nethercote, P. J. Stuckey, R. Becket, S. Brand, G. J. Duck, and G. Tack, *MiniZinc: Towards a Standard CP Modelling Language*. Springer Berlin Heidelberg, 2007, p. 529–543. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-74970-7_38

[59] I. Dunning, J. Huchette, and M. Lubin, "Jump: A modeling language for mathematical optimization," *SIAM Review*, vol. 59, no. 2, p. 295–320, Jan. 2017. [Online]. Available: http://dx.doi.org/10.1137/15M1020575

[60] B. Nicholson, J. D. Siirola, J.-P. Watson, V. M. Zavala, and L. T. Biegler, "pyomo.dae: a modeling and automatic discretization framework for optimization with differential and algebraic equations," *Mathematical Programming Computation*, vol. 10, no. 2, p. 187–223, Dec. 2017. [Online]. Available: http://dx.doi.org/10.1007/s12532-017-0127-0

[61] W. Mauerer and M. Schönberger, "A toolbox to understand the physics of quantum data management," in *Proceedings of the Third Workshop on Quantum Computing and Quantum-Inspired Technology for Data-Intensive Systems and Applications@IEEE SIGMOD*, ser. Q-Data '26, 5 2026.

[62] N. Quetschlich, L. Burgholzer, and R. Wille, "Towards an Automated Framework for Realizing Quantum Computing Solutions," in *International Symposium on Multiple-Valued Logic (ISMVL)*, 2023.

[63] D. Rovara, N. Quetschlich, and R. Wille, "A framework to formulate pathfinding problems for quantum computing," 2024. [Online]. Available: https://arxiv.org/abs/2404.10820