Quantum Data Encoding Patterns and their Consequences

Martin Gogeißl*
Technical University of Vienna
Vienna, Austria
martin.gogeissl@student.tuwien.ac.at

Hila Safi†
Siemens AG, Technology and
Technical University of Applied
Sciences Regensburg
Munich/Regensburg, Germany
hila.safi@siemens.com

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

ABSTRACT

The use of quantum processing units (QPUs) promises speed-ups for solving computational problems, in particular for discrete optimisation. While a few groundbreaking algorithmic approaches are known that can provably outperform classical computers, we observe a scarcity of programming abstractions for constructing efficient quantum algorithms. A good fraction of the literature that addresses solving concrete problems related to database management concentrates on casting them as quadratic unconstrained binary optimisation problems (QUBOs), which can then, among others, be processed on gate-based machines (using the quantum approximate optimisation algorithm), or quantum annealers. A critical aspect that affects efficiency and scalability of either of these approaches is how classical data are loaded into qubits, respectively how problems are encoded into QUBO representation. The effectiveness of encodings is known to be of crucial importance for quantum computers, especially since the amount of available qubits is strongly limited in the era of noisy, intermediate-size quantum computers.

In this paper, we present three encoding patterns, discuss their impact on scalability, and their ease of use. We consider the recreational (yet computationally challenging) Sudoku problem and its reduction to graph colouring as an illustrative example to discuss their individual benefits and disadvantages. Our aim is enable database researchers to choose an appropriate encoding scheme for their purpose without having to acquire in-depth knowledge on quantum peculiarities, thus easing the path towards applying quantum acceleration on data management systems.

KEYWORDS

Quantum computing, encoding strategies, encoding patterns

ACM Reference Format:
Martin Gogeißl, Hila Safi, and Wolfgang Mauerer. 2024. Quantum Data Encoding Patterns and their Consequences. In *Workshop on Quantum Computing and Quantum-Inspired Technology for Data-Intensive Systems and Applications (Q-Data '24), June 9, 2024, Santiago, AA, Chile.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3665225.3665446

*Work performed at Technical University of Applied Sciences Regensburg.
†Corresponding author.

Q-Data '24, June 9, 2024, Santiago, AA, Chile

ACM ISBN 979-8-4007-0553-3/24/06
https://doi.org/10.1145/3665225.3665446

1 INTRODUCTION

Quantum computing is a recent approach to computation that aims for solving problems faster than their classical counterparts. Few seminal examples like factoring large numbers [37, 39] or unstructured search [15] are proven to be more efficient than their classical counterparts. However, we are currently in the era of noisy intermediate-scale quantum (NISQ) computing, and algorithms designed for these machines face several challenges [4, 30]. For instance, quantum hardware is limited to a relatively small number of quantum bits (qubits), typically ranging from about 50 to 400. Scaling quantum computers to a larger numbers of qubits is a difficult engineering problem that depends on the specific hardware platform. Also, quantum computers are susceptible to noise and interference from their environment, and suffer from other imperfections that strongly affect algorithmic performance and capabilities [32]. While it is hard to predict the future development of foundational technologies, it is very likely that quantum computers will be inapt for handling large amounts of payload data [19]. In particular, information carriers with quantum properties are much less robust than any classical alternative, which by the very distinction between classical and quantum operates at larger physical scales. Classical variants are therefore more robust and easier to build. This might seem as a massive impediment towards the use of quantum technologies in database management systems (DBMS), and more generally for data processing tasks that require large amounts of input data. Yet, many problems in DBMS address issues that are independent from the actual payload data, and could therefore be amenable to quantum speedups.

Even if little amounts of data are involved, quantum algorithms that promise speed-ups often operate on the assumption that these can be efficiently loaded into quantum states. However, the available bandwidths are much smaller than in classical systems [19], and potential quantum speedups may easily be lost to such "practical" inefficiencies [50] that appear alongside problems on the quantum side [11]. In fact, the runtime complexity of the loading routine strongly depends on the selected data encoding, for which quantum bits offer substantially more flexibility than classical carriers of information, and the characteristics of the data. Encodings that use different degrees of freedom of a quantum bit allow us to specify how classical data are represented. Selecting a good encoding can significantly affect both, accuracy and scalability of the obtained solutions; improvements ranging over multiple orders of magnitude with improved problem encoding have been observed [46].

In this paper, we discuss three such encoding patterns, and analyse their impact on the seminal quantum algorithms based on quadratic, unconstrained binary optimisation problems which are

often used as base primitives for quantum-accelerated DBMS [5–7, 14, 16, 34–36]. To begin, we explore the most common encoding patterns and demonstrate their significant impact on our solutions. Building on this foundation, we can later investigate several other encoding patterns to determine the most appropriate one for specific problems [46].

By this, we extend the list of data encoding patterns and their influence on NISQ-era algorithms provided by Weingold *et al.* [46], which builds on Leymann's initial work on quantum software patterns [23]. To consistently illustrate the impact of the different patterns, we use two well-known combinatorial problems: Sudoku and graph colouring. Sudoku is a specific representative within the group of graph colouring problems. While both are not directly related to DBMS issues, they are widely known, we do not need to consider domain-specific peculiarities, but can directly focus on the effects of varying encoding methods. The obtained insights apply to a wide range of DBMS tasks (see the discussion of related work below), as the findings in terms of scalability and usability are independent of the underlying task.

Both problems are NP-complete, and therefore lack deterministic polynomial-time algorithms. Furthermore, graph colouring is an essential problem due to its wide range of applications in optimisation, parallel and distributed computing, circuit board design, scheduling, and various other fields. It serves as an important challenge for both algorithmic and engineering research. We provide a reproduction package [27] (link in PDF) that enables re-use and verification of our results.[1]

The rest of this paper is organised as follows. Section 2 reviews related work. In Sec. 3, we explain the principles behind our approach, and give definitions underlying the work. We continue with investigating different encoding strategies with the objective of gaining insights into their applicability in Sec. 4, and evaluate the encoding patterns using the graph colouring and Sudoku problem to asses their advantages and disadvantages in Sec. 5. We conclude in Sec. 6.

2 RELATED WORK

Using quantum approaches to accelerate DBMS has only recently started to receive attention, starting with initial work by Trummer and Koch on multi-query optimisation with quantum annealers [45]. Apart from that, transaction scheduling [5–7, 14], schema matching, or tuning index configurations [16] have been addressed using quantum methods. The problem of finding good orders in which to join table columns has also been cast as an optimisation problem by Schönberger *et al.* based on known transformations to mixed-integer linear programming [34], and using other encodings [35], including variants that allow for handling general *bushy* join trees by Nayak *et al.* [28] and Schönberger *et al.* [36]. Ref. [48] introduces a reinforcement learning inspired approach for the join order problem using variational quantum circuits. Winker *et al.* [49] review the available literature and classify potential use-cases, as likewise do Çalikyilmaz *et al.* [7] and Yuan *et al.* [52].

Leymann [23] laid the foundation for quantum algorithm pattern encoding. The paper introduces three pattern primitives and two data encoding patterns, and was subsequently extended with Weingold *et al.* [46] with six data-handling patterns. A systematic collection of patterns has been created by the PlanQK project, and is available as an interactive website. Solving Sudoku puzzles has been accomplished using Fujitsu's digital annealer, which is a quantum-inspired hardware accelerator for solving combinatorial optimisation problems. This study transforms the Sudoku problem utilising the one-hot encoding approach into a Quadratic Binary Optimisation (QUBO) formulation [22, 44]. To improve space efficiency, Tabi *et al.* [41] employed a binary encoding method for representing the colours in a graph colouring problem, deviating from the conventional one-hot encoding utilised in the QUBO formulation proposed by Lucas [25]. Besides an exponential reduction in circuit height in the number of colours and thus in required qubits, they also observed a decrease in the number of required layers and optimisation steps to attain optimal solutions. Several important and complex DBMS problems can be mapped onto the graph colouring problem, including, but not limited to, transaction and task scheduling problems [21], resource allocation problems [26], and deadlock prevention [51]. The effects of encoding methods for quantum annealing can be studied experimentally. Tamura *et al.* [42] explore performance variations of one-hot, unary, and binary encodings in solving the quadratic knapsack problem. Conducting experiments on the Digital Annealer, they observe that the inefficient unary encoding shows superior performance compared to binary encoding, and find that the one-hot encoding fails to achieve a feasible solution even when subjected to substantial penalising terms.

3 CONTEXT AND FOUNDATION

In this paper, we focus on NP-complete (NPC) problems, as this class contains practically relevant problems for DBMS that, assuming the usually uncontested P ≠ NP hypothesis, cannot be efficiently solved on a classical machine (most instances also hard to approximate, as is textbook knowledge [47]). A decision problem p is in NPC if a solution can be determined by a non-deterministic Turing machine in polynomial time (*i.e.*, $p \in$ NP), and is additionally NP-hard, which means that any other problem in NP can be reduced to p in polynomial time. In particular, we consider the Sudoku and graph colouring problem, as outlined before. Both are NP-complete combinatorial optimisation problems.

3.1 Sudoku

A game of Sudoku can be represented by an undirected graph $G = (V, E)$, where V is the set of cells in an $n^2 \times n^2$ Sudoku with $N = n^2$ and $|V| = N^2$. Cells that are in the same row, the same column or in the same sub-grid, are connected with an edge. The objective of a $N \times N$ grid is to find a numerical solution, where each row, column, and sub-grid contain all the numbers from 1 to N once. This can consequently be reduced to the graph colouring problem. An exemplary representation for an $n = 2$ Sudoku graph is shown in Fig. 1.

3.2 Graph Colouring

The graph colouring problem is a fundamental challenge in graph theory that finds numerous applications across a variety of fields including, but not limited to, biology, biochemistry, computer science, economics, electrical engineering, medicine, and network

[1] The material is also available at a long-term stable, DOI-safe location on Zenodo.

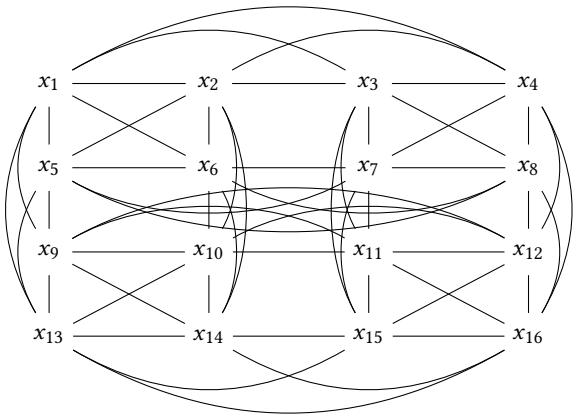

Figure 1: Sudoku instance with graph size 4×4.

analysis [43]. Formally, given a graph $G = (V, E)$, the task is to find a colouring function $c : V \to C$, where C is the set of colours, such that every edge $(u, v) \in E$, $c(u) \neq c(v)$. The goal is to minimise the size of C. This translates to assigning colours to the vertices of a graph in a way that ensures no two adjacent vertices share the same colour. The objective is to minimise the total number of required different colours.

4 ENCODING PATTERNS

Quantum algorithms, at their core, rely on input data provided via a suitable encoding into quantum states. Selecting an appropriate encoding method is essential to ascertain efficiency and accuracy of the quantum algorithm [46]. Multiple such strategies have been proposed in the literature. The three most common variants to encode binary numbers respectively bit-strings are summarised in Table 1: We consider how to map a bit-string $x = x_0 x_1 \dots x_{D-1}$ to a corresponding number within the Sudoku grid $I = \{1, 2, \dots N\}$. While this might seem a relatively straightforward task at a first glance, we will discuss below that the choice comes with a number of consequences.

Table 1: Common encoding options and resource requirements D for an integer value I, applied to the bit-string x.

Encoding Pattern	Encoding	Req. qubits D
One-Hot	$I \mapsto 1 + \sum_{i=0}^{D-1} i \cdot x_i$	$D = N$
Unary	$I \mapsto 1 + \sum_{i=0}^{D-1} x_i$	$D = N - 1$
Binary	$I \mapsto 1 + \sum_{i=0}^{D-1} 2^i \cdot x_i$	$D = \lceil \log_2 N \rceil$

Apart from that, we consider two different encoding strategies that allow us to map higher-level structural information about problems into quantum states: For one, based on *standards graph* that comprise nodes connected by edges, are ubiquitous in computer science. On the other hand, we consider *hypergraphs*, in which one single edge can connect to an unlimited number of vertices.

4.1 Standard Graph Strategies

The initial phase of creating an encoding scheme for Sudoku involves selecting an appropriate binary-integer encoding. The decision affects the potential constraints and penalty functions that can be implemented to penalise invalid Sudoku states. Standard graph strategies apply to all standard graph colouring problems. The main focus of this function is to compare two adjacent nodes and apply a penalty if they contain the same value. If they do not, the output of the function will be 0. A graph colouring is valid if the sum of all comparisons vanishes. As described in Section 3.1, Sudoku can be reduced to a graph colouring problem.

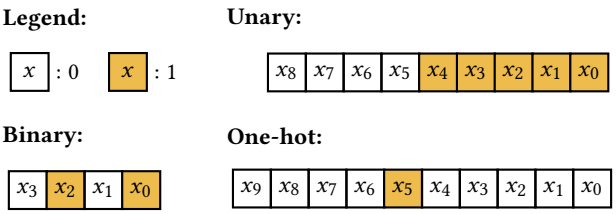

Figure 2: Binary, unary, and one-hot encoding for integer value $I = 5$.

4.1.1 Comparison-Based One-Hot Encoding. This encoding is based on the Ising formulation for graph colouring problems, which was introduced by Lucas [25]. It consists of two rules:

(1) Every node v is assigned exactly one number:

$$\sum_{i=0}^{D-1} x_{v,i} = 1, \forall v \in V \tag{1}$$

(2) Two adjacent nodes in the Sudoku graph cannot share the same number:

$$\sum_{i=0}^{D-1} x_{u,i} x_{v,i} = 0, \forall (uv) \in E \tag{2}$$

The first constraint ensures that every bit-string has exactly one 1 to comply with the one-hot encoding scheme visualised in Fig. 2. The second constraint prohibits setting a 1 in the same position of the bit-string for two adjacent nodes. The combination of these two constrains results in the first penalty function, which is 0, if all constraints are satisfied and $H_P > 0$, if the Sudoku is in an invalid state.

Despite Lucas [25] advocating for retaining positive constants in certain QUBO formulations to aid in conceptual clarity and separate energy scales, Glover *et al.* [13] took a different approach. They assert that, in the absence of an explicit objective function, any positive value for the penalty suffices for finding a feasible colouring within the allowed colours. This shift renders the traditional penalising coefficient unnecessary in the formulations proposed in this paper. The combined sum of both constraints detailed above result in the final problem Hamiltonian

$$H_P = \sum_{v \in V} \left(1 - \sum_{i=0}^{D-1} x_{v,i} \right)^2 + \sum_{(uv) \in E} \sum_{i=0}^{D-1} x_{u,i} x_{v,i}. \tag{3}$$

4.1.2 Comparison-Based Unary Encoding. The Hamming weight determines the integer value of a unary encoded bit-string. Therefore, if the bit counts are unequal, two bit-strings have different values. Unlike one-hot encoding, this approach takes into account the bit sum, ignoring their position in the bit-string. Moreover, an additional bit to represent the number 0 is not needed since this value is indicated by an all zero bit-string. A n-bit unary number can correspond to $n+1$ different integer values. As a result, two neighbouring cells x_v and x_u are different if the inequality

$$\sum_{i=0}^{D-1} x_{u,i} \neq \sum_{i=0}^{D-1} x_{v,i}, \forall (uv) \in E \tag{4}$$

holds true.

This results in a penalty function that uses slack variables to transform it into an equality function [13],

$$H_P = \sum_{(uv)\in E} \left(- \left(\sum_{i=0}^{D-1} x_{u,i} - x_{v,i} \right)^2 + 1 + s \right)^2, \tag{5}$$

with s being the binary expansion of a slack variable, minimising the expression:

$$s = \sum_{i=0}^{\lfloor \log(D^2-1) \rfloor} 2^i s_i, \forall s_i \in \mathbb{B}. \tag{6}$$

4.1.3 Binary Encoding. In terms of information, the binary encoding method is the densest representation of an integer value. Therefore, in theory, the number of qubits required for computation can be reduced. Similar to the one-hot encoding method, binary encoding may require a restriction to limit its value range. If the number of integers in the value domain n is not a power of 2, there may be invalid numbers between n and the next greatest power of 2. To avoid this possibility, preventative measures should be taken. For example, a 9×9 Sudoku requires four bits to represent the numbers from 1 to 9, but a four bit binary encoding scheme can represent 16 different values, of which the numbers from ten to 16 are considered invalid in a standard Sudoku. Because of their quadratic nature, this limitation does not apply to all Sudoku problems sizes. If the size of the Sudoku is a power of two, the number of different values needed to solve the Sudoku and the value range of the corresponding binary number are the same, thus no value limitation is necessary. There are approaches to restrict the value domain and here are a few examples:

(1) Redefining the integer encoding function
(2) Inequality constraints
(3) Boolean expression penalties

(1) refers to the adjustment of the integer-to-binary coding function to assign all binary number values above the threshold to an integer below the threshold. The revised function is defined as [25]:

$$I \mapsto \left(\sum_{i=0}^{D-2} 2^i x_i \right) + (N + 1 - 2^{D-1}) x_{D-1}. \tag{7}$$

Applied to the 9×9 Sudoku case, the integer values between nine and 15 are mapped to valid numbers between three and eight.

The second point introduces an additional inequality constraint, which penalises values exceeding the threshold. We construct the formula as follows:

$$1 + \sum_{i=0}^{D-1} 2^i x_i \leq N. \tag{8}$$

With slack variables, the inequality is transformed to

$$\sum_{i=0}^{D-1} 2^i x_i - N + 1 + s = 0, \text{with } s = \sum_{i=0}^{\lfloor \log_2(N-1) \rfloor} 2^i s_i, \forall s_i \in \mathbb{B}, \tag{9}$$

which leads to penalty

$$H_1 = \sum_{v \in V} \left(\sum_{i=0}^{D-1} 2^i x_{v,i} - N + 1 + \sum_{i=0}^{\lfloor \log_2(N-1) \rfloor} 2^i s_i \right)^2. \tag{10}$$

Point (3) is to set a SAT constraint for the bit sequence, whose satisfaction depends on whether the bit-string is exists in an approved state or not. This can be constructed manually by creating a formula in disjunctive normal form that concatenates all valid states and then minimises the expression. Alternatively, an algorithm can aid in constructing a polynomial that penalises all invalid forms [12]. Based on the value of the bit, the algorithm combines decision variables using logical *and* and *or* expressions. If the value is one, subsequent bits are added with a logical *and*, otherwise with a logical *or*. The process is performed in descending bit order until the last zero of the bit-string is reached.

Table 2: Boolean expressions to restrict the value set.

Size	max I_{10}	max I_2	Boolean Expression
9×9	8	1000	$x_3 \wedge (x_2 \vee x_1 \vee x_0)$
25×25	24	11000	$x_4 \wedge x_3 \wedge (x_2 \vee x_1 \vee x_0)$
36×36	35	100011	$x_6 \wedge (x_5 \vee x_4 \vee x_3)$

The expressions presented in Table 2 result in a penalty greater than zero if the clauses are satisfied. To map between logical and mathematical operations, we substitute *and* with a multiplication and *or* with addition. In contrast, the conventional conversion method to addition requires an additional $-x_1x_2$ term to normalise the expression to 1. Since our penalty function does not necessitate normalisation, we can omit this term.

Table 3: Penalty terms to ensure value restrictions.

Base	Size	max I_2	Penalty
3	9×9	1000	$x_3 \cdot (x_2 + x_1 + x_0)$
5	25×25	11000	$x_4 \cdot x_3 \cdot (x_2 + x_1 + x_0)$
6	36×36	100011	$x_5 \cdot (x_4 + x_3 + x_2)$

For each vertex in the Sudoku graph, the terms listed in Table 3 are added to the penalty function. The goal is to ensure that the maximum value does not exceed a specific threshold. If a node should receive a value above this threshold during the calculation process, a penalty constant greater than zero is added.

The purpose of this initial phase was to constrain the potential values due to the variance between problem size and the binary

value range. Subsequently, a comparison is necessary to avoid the occurrence of two identical binary numbers.

$$H_2 \begin{cases} = 0 & \text{if } x_u \neq x_v \\ > 0 & \text{if } x_u = x_v \end{cases} \tag{11}$$

Penalty functions can be formed in multiple ways, from basic yet inefficient concepts to more streamlined and optimised approaches. Comparing the difference of converted integers and utilising slack variables in inequalities demands a significant amount of additional variables and considerable effort. A penalty function based on the integer representation is given by

$$f(I, I') = (-(I - I')^2 + 1 + s)^2, \tag{12}$$

and can be derived akin to the unary approach in Eq. (4) from the inequality $I \neq I'$ of two binary encoded value sums Eq. (7). Following the same steps, this inequality is transformed into $(I - I')^2 > 0$ and finally by applying the slack variable trick into the penalty function above, which provides the desired boolean decision output of either 0 of 1. The slack variable s should be set to cover the whole size of N^2, which results in inefficient use of the integer expansion.

An alternative approach is to compare single bits instead of the computed integer value. This requires an injective binary to integer function, leading to the implementation of either value restriction method (2) or (3). Two (essentially identical) methods can accomplish this task, consisting of either a disjunctive concatenation given by

$$f(x, y) = \neg((x_0 \oplus y_0) \vee (x_1 \oplus y_1) \vee \ldots \vee (x_{D-1} \oplus y_{D-1})), \tag{13}$$

or a conjunctive concatenation derived from applying a transformation with De Morgan's law:

$$f(x, y) = \neg(x_0 \oplus y_0) \wedge \neg(x_1 \oplus y_1) \wedge \ldots \wedge \neg(x_{D-1} \oplus y_{D-1}). \tag{14}$$

Both Boolean expressions evaluate to true if vectors x and y are identical, and return false if at least one bit differs. False in this case means zero, which is the desired outcome of the penalty function for unequal binaries. For a true result, a number above zero is obtained.

Table 4: QUBO expressions for elementary Boolean logical operations.

Operation	QUBO expression	Operation	QUBO expression
$\neg x_1$	$-x_1 + 1$	$x_1 \oplus x_2$	$-2x_1x_2 + x_1 + x_2$
$x_1 \vee x_2$	$-x_1x_2 + x_1 + x_2$	$\bigwedge_{i=1}^{k} x_i$	$\prod_{i=1}^{k} x_i$
$x_1 \wedge x_2$	x_1x_2	$\bigvee_{i=1}^{k} x_i$	$1 - \prod_{i=1}^{k}(-x_1 + 1)$

To cast Boolean expressions in QUBO form, we can rely on established methods [10, 17, 40]; the transformations required for our purpose are summarised in Table 4. Converting Eq. (13) respectively Eq. (14) delivers the desired comparison function:

$$f(x, y) = \prod_{i=0}^{D-1} (2x_iy_i - x_i - y_i + 1) \tag{15}$$

The information dense binary encoding necessitates a linkage of all decision variables within two comparable bit-strings. If this cannot be ensured, then the minimised energy function outputs not any two different bit-string, but rather the two *most different* bit-strings. This side effect introduces an unwanted bias, which for the case of Sudoku implies that the probability of an eight next to a nine is smaller than the probability of a one next to a nine, whereas the desired outcome is a constant probability. Converted to a penalty function, this linkage is achieved by higher-order polynomials. Those can be implemented directly onto certain quantum algorithms like QAOA [12], or can be reduced to quadratic models preemptively by applying polynomial reduction methods [13, 33]. Depending on the conversion method, circuit width may increase with the introduction of additional auxiliary variables, and circuit depth may likewise increase given the need to apply additional constraints. However, automatic conversion methods [24] will eventually provide optimal conversions for a given set of requirements.

The final Hamiltonian is obtained by merging a value restriction method H_1 with Eq. (15), resulting in

$$H_P = H_1 + \sum_{(uv) \in E} \prod_{i=0}^{D-1} (2x_{u,i}x_{v,i} - x_{u,i} - x_{v,i} + 1)). \tag{16}$$

Another approach to identifying equal binaries is to sum up the connected *xor* expressions rather than linking them with *or* gates by

$$\sum_{i=0}^{D-1} x_i \oplus y_i \geq 1, \tag{17}$$

which together with the previously discussed slack variable allows us to construct the transformation

$$H_P = H_1 + \sum_{(uv) \in E} \left(\sum_{i=0}^{D-1} 2x_{u,i}x_{v,i} - x_{u,i} - x_{v,i} + 1 + \sum_{i=0}^{\lfloor \log(D-1) \rfloor} 2^i s_i \right)^2. \tag{18}$$

4.2 Hypergraph Strategies

Let us now commence by further improving the derivation method. Instead of comparing each individual vertex pair within a row, we add up all i-bits in a row, column or sub-grid, respectively. This approach can be represented as a *hypergraph* (recall that an edge can connect more than two vertices in such a graph, as illustrated in Fig. 3). Rather than examining each distinct node pair separately, this approach allows us to compare an entire row, column, or sub-grid simultaneously. Implementing the method necessitates selecting an unambiguous encoding scheme for the node values, which allows us to add all coordinate bits to a point where equivalent node values are avoided. For this reason, only one-hot and unary encoding schemes are viable.

While the hypergraph encoding provides a visual intuition for the one-hot encoding, it is identical to the standard graph variant in terms of implementation. Conversely, the unary hypergraph-based encoding does offer a computational advantage compared to the former unary encoding, which is solely based on the inequality of the Hamming weight of two bit-strings. It is possible to calculate the sum of bits of equal arity and eliminate duplicate node values.

4.2.1 Hypergraph One-Hot Encoding. The hypergraph one-hot encoding corresponds to the solution implemented in the Topcoder competition [22, 44]. The Sudoku is represented as a $N \times N \times N$ cube, wherein the 1-bits are arranged in a specific manner. This

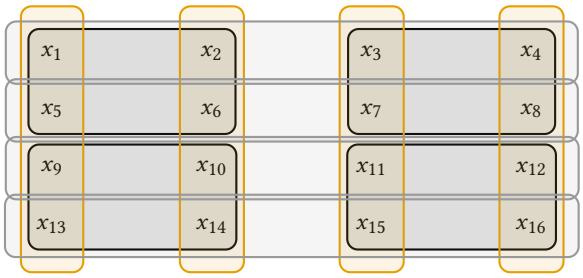

Figure 3: Representing a Sudoku instance as a hypergraph.

ensures that each row segment, column segment, layer segment and $n \times n$ sub-grid has exactly one single bit set.

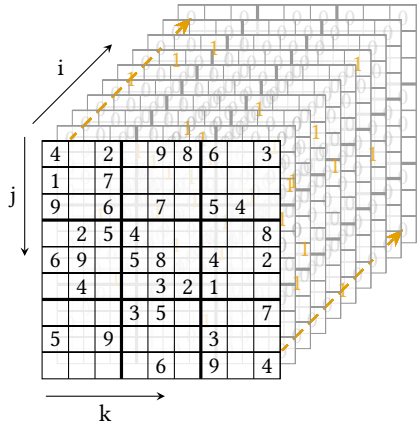

Figure 4: A $9\times9\times9$ cube representation of a bit- based Sudoku. The front $x_{0,j,k}$ (added as visual aid) forms the visible Sudoku problem, and the vector $x_{i,j,k}$ (with j,k constant) gives the value of each populated cell $x_{j,k}$ in one-hot encoding. Two examples for such vectors are given by dashed yellow lines for cells $x_{0,1,1}$ and $x_{0,9,9}$.

In the 3D model depicted in Fig. 4 a solution is valid if each row, column, depth vector and i-layer sub-grid comprises exactly one 1. Based on this, we establish the following constraints:

(1) Each node can only hold a single value:

$$H_1 = \sum_{j,k} \left(\sum_{i=0}^{D-1} x_{j,k,i} - 1 \right)^2 .$$

(2) Each column k cannot have a duplicate number:

$$H_{\text{col}} = \sum_{i=0}^{D-1} \sum_{k} \left(\sum_{j} x_{j,k,i} - 1 \right)^2 .$$

(3) Each row j cannot have a duplicate number

$$H_{\text{row}} = \sum_{i=0}^{D-1} \sum_{j} \left(\sum_{k} x_{j,k,i} - 1 \right)^2 .$$

(4) Each of the N $n\times n$ sub-grids cannot have a duplicate number

$$H_{\text{sub}} = \sum_{i=0}^{D-1} \left(\sum_{j}^{n-1} \sum_{k}^{n-1} x_{(j+s),(k+t),i} - 1 \right)^2$$

$$s,t \in \{0, n, 2n, \ldots, N-n\} : \textit{offset to each grid}$$

The final Hamiltonian is then composed as

$$H_P = H_1 + H_{\text{col}} + H_{\text{row}} + H_{\text{sub}}. \tag{19}$$

The quadratic terms align with the result from the one-hot encoding method presented in Section 4.1.1. The difference lies in the problem-solving approach.

4.2.2 Hypergraph Unary Encoding. Similar to the one-hot encoding, the hypergraph-based unary encoding adds up an entire row rather than comparing two elements within a row for inequality. This method requires one bit less than one-hot encoding, leading to a theoretical advantage of $\frac{1}{N}$ less space consumption. To achieve this, some restrictions must be introduced. As exemplified

Table 5: Unrestricted (left) and restricted (right) format for bit-strings (BS).

BS	Bit position				Dec.
	3	2	1	0	value
x_0	1	0	1	0	2
x_1	1	1	1	1	4
x_2	0	0	0	0	0
x_3	1	1	0	1	3
x_4	0	1	0	0	1
Σ	3	3	2	2	

BS	Bit position				Dec.
	3	2	1	0	value
x_0	0	0	1	1	2
x_1	1	1	1	1	4
x_2	0	0	0	0	0
x_3	0	1	1	1	3
x_4	0	0	0	1	1
Σ	1	2	3	4	

in Table 5, a sum of equal valued bits that yields a constant and predictable result, is only possible if the bit order is fixed. Determining a constraint that is only satisfied if the bit-string conforms with the format $0^a 1^{D-a}$ with $0 \le a \le D$ can be derived from the fact that a set bit 1 must not be followed by an unset bit 0.

Table 6: Truth table for the unary form restriction.

x_i	x_{i+1}	f
0	0	0
0	1	0
1	0	1
1	1	0

The equivalent Boolean expression $x_i \wedge \neg x_{i+1}$ can be derived from the truth table presented in table Table 6. A true value indicates a constraint violation, and the assigned penalty function returns 1. Transformed to a penalty function, the following statement satisfies the requirements:

$$H_1 = \sum_{v \in V} \sum_{i=0}^{D-2} x_{v,i} - x_{v,i} x_{v,i+1} \tag{20}$$

With the established structure, we can now create a sum over all identical valued bits within each row, column, and sub-grid. This

cumulative sum is intended to match the current index of the summation. This differs from one-hot encoding, where the sum across each section is expected to be 1. The differences in the pattern restriction and the hypergraph edge comparison are depicted in Table 7. The constraints apply for each vertex, and for each edge e of the set of hypergraph edges E, respectively.

Table 7: Differences in constraints for one-hot and unary encoding.

Constraint	One-Hot	Unary
Pattern Adherence $\forall v \in V$	$\sum_{i=0}^{D-1} x_{v,i} = 1$	$\sum_{i=0}^{D-2} x_{v,i} - x_{v,i}x_{v,i+1} = 0$
Value Distinctness: $\forall i \in \{0,\dots,D-1\}$, $\forall e \in E$	$\sum_{v\in e} x_{v,i} = 1$	$\sum_{v\in e} x_{v,i} = i+1$

In the unary objective functions, the present index i is subtracted to minimise the function when the sum equals i. The highlighted term i deviates from the one-hot encoding scheme:

(1) Each column k can not have a duplicate number

$$H_{\text{col}} = \sum_{i=0}^{D-1} \sum_{k} \left(\sum_{j} x_{j,k,i} - i - 1 \right)^2 .$$

(2) Each row j can not have a duplicate number

$$H_{\text{row}} = \sum_{i=0}^{D-1} \sum_{j} \left(\sum_{k} x_{j,k,i} - i - 1 \right)^2 .$$

(3) Each of the N $n\times n$ sub-grids cannot have a duplicate number

$$H_{\text{sub}} = \sum_{i=0}^{D-1} \left(\sum_{j}^{n-1} \sum_{k}^{n-1} x_{(j+s),(k+t),i} - i - 1 \right)^2$$

where $s, t \in \{0, n, 2n, \dots, N-n\}$ denote the offset to each grid.

This results in the final Hamiltonian for the hypergraph based unary encoding that is given by

$$H_P = H_1 + H_{\text{col}} + H_{\text{row}} + H_{\text{sub}}. \tag{21}$$

5 METHOD COMPARISON

Comparing the qualities of the encodings is possible based on different criteria. Unfortunately, estimating practical performance in terms of execution time and result quality is—at the current stage of development in the NISQ era—impossible without considering peculiarities of the target hardware. Given the multitude of approaches ranging from trapped ions and neutral atoms via semiconductor-based implementations to photonic approaches, we restrict our analysis to three proxy quantities that allow for deriving concrete performance measures given sufficient details of the target hardware: Circuit depth and width, and the use of two-qubit gates that are known to dominate the introduction of errors into quantum computations. All metrics can, for the logical circuits, be directly derived from the Hamiltonians, albeit it should be taken into account that compilation from logical to physical circuits will introduce further inefficiencies. As the transformation process is still subject of intensive research (*e.g.*, Refs. [29, 31]), we cannot go into more details in this aspect.

However, considering specific values for $n = 2$ and $n = 3$ for circuit depth and two-qubit gates shown in Table 9. For $n = 3$, only the best performing results are calculated, as the large amount of required qubits in the standard unary case drastically impacts the time required to generate the ansatz. Circuits heights are summarised in Table 8 for the different approaches. We discuss and evaluate our findings in the following.

5.1 Complexity Metrics

5.1.1 Circuit Height. The hypergraph encoding methods and the standard graph one-hot method do not require additional variables (*e.g.*, slack variables) and demonstrate a better scaling behaviour with increasing problem sizes. This can be attributed to the dense graph structure inherent in Sudoku problems. The number of edges in the graph, coupled with the addition of variables per edge, grows at a much higher rate than the number of qubits required to represent the Sudoku grid. Although binary inequality has a quadratic scale advantage in their standard variables count, they exhibit an overall poorer scaling behaviour. The higher number of standard variables for hypergraph encoding methods and standard graph one-hot encoding is, in comparison, not relevant. However, for other problems, this method may be a more suitable option. For instance, Tabi *et al.* [41] demonstrate that the necessary circuit widths for solving a particular graph colouring problem can be exponentially reduced in the number of colours, while decreasing the number of layers and optimisation iterations for a QAOA execution to reach an optimal solution.

5.1.2 Circuit Width and Two-Qubit Count. Both circuit width and the number of required two-qubit gates rely heavily on the ansatz generator and the optimisations that are possible by following standard job-shop scheduling heuristics [20]. The substantial differences in these metrics between the strategies is shown by a QAOA implementation of the encodings for $n = 2$ in Table 9. The methods *Default*, *Coloring* and *Gray Synth* refer to options offered by the *Qaptiva* framework [2, 3] to synthesise and optimise the QAOA ansatz. "Default" corresponds to implementing the default ordering provided by the input observable. "Coloring" applies graph colouring techniques to order the terms, aiming at reduced circuit depth. The third strategy, "Gray Synth", employs an algorithm by Amy *et al.* [1] that intends to minimise the number of CNOT gates through heuristic methods inspired by Gray codes. This advantage is expected to be most prominent for QAOA instances with large clauses.

The standard unary encoding is a notable outlier, as the substantial number of required slack variable impacts the amount of gates and consequently circuit depth. The hypergraph strategy shows best universal performance, and uses the least amount of space. As expected, both binary variants lead to the longest circuits, as a drawback of the $\log N$ qubit amount. The potential reduction of CNOT gates through the Gray Synth approach materialises only for the binary boolean encoding. This is, because unlike other encodings, it uses higher- order polynomials with up to $2 \log N$ variables.

Table 8: Amount of qubits required for each encoding method (omitting rounding functions for logarithms for the sake of simplicity) for Sudoku instances of size $n \times n$ with $N = n^2$.

Representation	Encoding	Decision variables	Slack variables per edge	Total amount of qubits
Standard Graph	One-hot	N^3	-	N^3
	Unary	$N^2(N-1)$	$1+\log(N^2-2N)$	$N^2(N-1)+0.5N^2(3N-2n-1)(1+\log(N^2-2N))$
	Binary Bools	$N^2\log(N)$	-	$N^2\log(N)$
	Binary Ineq.	$N^2\log(N)$	$1+\log(\log(N)-1)$	$N^2\log(N)+0.5N^2(3N-2n-1)(1+\log(\log(N)-1))$
Hyper-graph	One-hot	N^3	-	N^3
	Unary	$N^2(N-1)$	-	$N^2(N-1)$

Table 9: Circuit Depth and CNOT gate count for $n = 2, 3$

	$n=2$						$n=3$	
	Default		Coloring		Gray Synth		Coloring	
Encoding	Depth	#CNOT	Depth	#CNOT	Depth	#CNOT	Depth	#CNOT
One-hot	169	640	33	640	1423	1571	101	20,412
Unary	9325	20,256	1501	20,256	40,619	39,460	-	-
Binary Booleans	296	560	72	560	464	526	1872	79,380
Binary Inequality	527	1008	107	1008	836	1188	850	63,180
Hypergraph One-hot	169	640	33	640	1423	1571	101	20,412
Hypergraph Unary	39	400	29	400	499	801	89	14,094

For the case of $n = 3$, the results in Table 9 detail the value for the colouring strategy. If compared to the $n = 2$ case, the increase in depth and CNOT gate count is observable. While the depth increases by a factor of three for the unary and one-hot cases, it increases by a factor of 26 for the binary boolean case and eight for the binary inequality case. The table reveals the poorer depth scaling behaviour of the binary encodings, which is a downside of requiring less qubits to encode the problem.

To select the most appropriate encoding for a specific problem, we find that currently, there is no substitute for awareness of details of various possible choices. The advantages and disadvantages of the encoding strategies discussed in this paper are summarised in Table 10.

Table 10: Advantages and disadvantages of the subject encodings.

Encoding	Advantages	Disadvantages
One-Hot	+ Simple + Universal + Hamming weight	- Space-intensive - Strict pattern
Unary	+ Intuitive + No arity	- Space-intensive - No arity: slacks required
Binary	+ Information dense + Space efficient + BCD variants	- Value restriction - Difficult to set constraints - Deeper Circuits

5.2 Discussion

5.2.1 One-Hot Encoding. One-Hot Encoding is a common and universally applicable technique implemented in addressing combinatorial optimisation problems. It is a straightforward design approach and does not strongly depend on multi-qubit gates. This can potentially simplify mapping logical problem descriptions to physical hardware graphs. Furthermore, one-hot encoding is known to provide better fault tolerance than binary encoding [8, 18]. This not only makes it advantageous for optimisation problems, but also highlights its suitability for quantum machine learning applications, particularly in managing random discrete variables [38]. The value of a one-hot encoded bit string is solely determined by the position of the single 1 within the bit-string, hence its Hamming weight is fixed. This ensures uniformity and can be beneficial for error detection and correction. However, one-hot encodings have low information density and require more decision variables to represent certain problems, as summarised in Table 8. The scalability and performance of one-hot encoding for larger problems is limited by the amount of available qubits, and presents a challenge for conducting experiments to empirically assess its effectiveness on current hardware [8].

In the case of large problems, approaches based on one-hot encoding may struggle to attain a viable solution because it fails to adhere to the one-hot pattern restriction [42]. Following this pattern for $N \to \infty$, the likelihood of a bit being set to 1 diminishes, hindering the optimisation process significantly. Additionally, as the compliance with the one-hot pattern becomes increasingly difficult, the need for a larger penalty factor arises. Given that the

objective function is minimised when a single decisive bit is correctly set, the probability of such an occurrence approaches zero as N approaches infinity.

5.2.2 Unary Encoding. We perceive this encoding to be intuitive, as it eliminates the need for positional information, which simplifies the representation of data. Each number is uniquely identified by the count of ones, providing a clear and concise encoding without the necessity for additional background information about the grid structure. Additionally, the number of ones + 1 in the representation directly corresponds to the value of the Sudoku entry. One significant advantage of the unary hypergraph strategy is that, compared to one-hot encoding, it requires fewer interrelated qubits, as can be seen from how the bit-string is constrained: For one-hot encoding, each qubit is connected to every other qubit in the same bit-string through a quadratic term. This results in $N(N-1)/2$ point-to-point connections. However, for unary encoding, each qubit is only connected to its neighbouring qubits, which only requires $N-2$ connections [9]. The implications of this benefit on a specific instance are shown in Table 9, where the hypergraph unary encoding requires 37.5% less CNOT gates and its circuit is between 12% to 77% more shallower when compared to the One-Hot encoding. If the bit-string regulation is omitted, no arity is required, but this benefit comes at the price of transforming into an inequality function and adding slack variables, hence increasing circuit height. This increases the probability of encountering errors [32]. Unary encoding has a low information density. Moreover, with increasing problem size, the advantage of unary encoding over one-hot encoding diminishes, as the benefit of saving a single qubit becomes irrelevant.

5.2.3 Binary encoding. This encoding is notable for its space efficiency, as it requires the fewest number of qubits for representing a problem, and exhibits a high information density.

Moreover, a certain degree of versatility apart from the standard binary model is possible. Alternative systems, each with different bit arities and bit-string to integer mappings, provide flexibility and adaptability depending on the use case. An example is Gray code, which showcases a distinctive sequence where adjacent values have only one differing bit, yielding benefits for applications that respond to incremental changes. This flexibility in selecting from a variety of binary encodings, all with the same bit length, provides designers with the ability to customise representations to meet specific needs.

However, it is necessary to impose an additional constraint to restrict the value range of binary encoded problems. Invalid numbers may exist between n and the next greatest power of 2, if n is not already such, as we have discussed in Section Section 4.1.3. Further drawbacks include the difficulty of defining objective functions, which is due to its high information density. Binary encoding enables the representation of the same amount of information with $\log(n)$ bits, whereas one hot encoding, for example, uses N bits. Furthermore, due to the introduction of higher polynomials when formulating constraints, binary encoding has resulted in a deeper circuit depth for our examples, increasing the overall computational complexity [12].

6 OUTLOOK AND CONCLUSION

We have presented different encoding strategies that allow us to represent the general Sudoku problem as a binary optimisation problem, and have also studied different derivations of the one-hot, unary and binary encoding using the graph colouring problem representation of Sudoku. We have considered a standard graph representation, and a hypergraph-based strategy. The simplicity of the hypergraph approach is useful to gain a better understanding on the connection different cells have within a row, while the standard graph approach correlates to the common understanding of graphs with adjacent cell restrictions.

We have provided quantitative estimates on resource use for all encodings, and have shown that the choice of encoding can negatively effect scalability of quantum algorithms by deteriorating space requirements and computational efficiency. Encoding methods for Sudoku that require extra slack variables have a more adverse effect on scaling behaviour than starting with a larger variable count. As the number of slack variables depends on the number of edges, this outcome may vary when dealing with problem graphs that have fewer connections. There are potential improvements to the formulated Hamiltonians. Some of them can be adapted with small changes, whereas others require an extensive reconstruction. In our setup, inequalities have been solved using slack variables. This advantage of this method is the creation of an unambiguous function that computes to 1, if $x_u = x_v$ and to 0, if $x_u \neq x_v$. Although this is an optimal constraint result, the disadvantage is that the inclusion of many additional variables adds substantially to space complexity. One potential solution for overcoming this limitation is to transform the inequality constraint into an inequality objective function. This alters the task from determining whether two bit-strings are unequal to identifying the two bit-strings that are the most dissimilar.

Our results highlight that the choice of encoding can substantially reduce the number of qubits required for encoding problems, yet may comes at the expense of (substantially) increased circuit depth. Depending on the available target hardware, the properties of gate operations like quality or speed, and the amount of available quantum bits may vary substantially, unlike with classical systems. Carefully considering the actually desired properties of potential quantum solutions is also required. We believe that our comparison provides a useful resource that helps database researchers to start with finding appropriate encoding strategies, and allows them to focus more on DBMS peculiarities instead of quantum details.

ACKNOWLEDGMENTS

This work was supported by the German Federal Ministry of Education and Research (BMBF), within the funding program 'quantum technologies—from basic research to market', grant numbers 13NI6092 (WM, MG) and 13N16093 (HS). WM acknowledges support by the High-Tech Agenda of the Free State of Bavaria.

REFERENCES

[1] Matthew Amy, Parsiad Azimzadeh, and Michele Mosca. 2018. On the controlled-NOT complexity of controlled-NOT–phase circuits. *Quantum Science and Technology* 4, 1 (Sept. 2018), 015002. https://doi.org/10.1088/2058-9565/aad8ca

[2] Atos. 2023. myQLM - Quantum Computing Framework. Retrieved 2024-05-31 from https://atos.net/en/lp/myqlm

[3] Atos. 2023. Quantum Learning Machine. Retrieved 2024-05-31 from https://atos.net/en/solutions/quantum-learning-machine

[4] Kishor Bharti, Alba Cervera-Lierta, Thi Ha Kyaw, Tobias Haug, Sumner Alperin-Lea, Abhinav Anand, Matthias Degroote, Hermanni Heimonen, Jakob S. Kottmann, Tim Menke, Wai-Keong Mok, Sukin Sim, Leong-Chuan Kwek, and Alán Aspuru-Guzik. 2022. Noisy intermediate-scale quantum algorithms. *Rev. Mod. Phys.* 94 (Feb 2022), 015004. Issue 1. https://doi.org/10.1103/RevModPhys.94.015004

[5] Tim Bittner and Sven Groppe. 2020. Avoiding blocking by scheduling transactions using quantum annealing. In *Proc. of the 24th Symposium on Int. Database Engineering & Applications* (Seoul, Republic of Korea) *(IDEAS '20)*. Association for Computing Machinery, New York, NY, USA, Article 21, 10 pages. https://doi.org/10.1145/3410566.3410593

[6] Tim Bittner and Sven Groppe. 2020. Hardware Accelerating the Optimization of Transaction Schedules via Quantum Annealing by Avoiding Blocking. *Open Journal of Cloud Computing (OJCC)* 7, 1 (2020), 1–21.

[7] Umut Çalikyilmaz, Sven Groppe, Jinghua Groppe, Tobias Winker, Stefan Prestel, Farida Shagieva, Daanish Arya, Florian Preis, and Le Gruenwald. 2023. Opportunities for Quantum Acceleration of Databases: Optimization of Queries and Transaction Schedules. *Proc. VLDB Endow.* 16, 9 (2023), 10 pages. https://doi.org/10.14778/3598581.3598603

[8] Bingren Chen, Hanqing Wu, Haomu Yuan, Lei Wu, and Xin Li. 2022. A Logarithm Depth Quantum Converter: From One-hot Encoding to Binary Encoding. arXiv:2206.11166 [quant-ph]

[9] Philippe Codognet. 2022. Domain-Wall / Unary Encoding in QUBO for Permutation Problems. In *2022 IEEE International Conference on Quantum Computing and Engineering (QCE)*. IEEE Computer Society, Los Alamitos, CA, USA, 167–173. https://doi.org/10.1109/QCE53715.2022.00036

[10] D-Wave. 2023. Reformulating a Problem — D-Wave System Documentation. Retrieved 2023-02-14 from https://docs.dwavesys.com/docs/latest/handbook_reformulating.html#elementary-boolean-operations

[11] Maja Franz, Lucas Wolf, Maniraman Periyasamy, Ch. Ufrecht, D. Scherer, A. Plinge, Ch. Mutschler, and Wolfgang Mauerer. 2023. Uncovering instabilities in variational-quantum deep Q-networks. *Journal of the Franklin Institute* 360, 17 (2023), 13822–13844. https://doi.org/10.1016/j.jfranklin.2022.08.021

[12] Adam Glos, Aleksandra Krawiec, and Zoltán Zimborás. 2022. Space-efficient binary optimization for variational quantum computing. *npj Quantum Information* 8 (04 2022), 39. https://doi.org/10.1038/s41534-022-00546-y

[13] Fred Glover, Gary Kochenberger, and Yu Du. 2019. A Tutorial on Formulating and Using QUBO Models. arXiv:1811.11538 [cs.DS]

[14] Sven Groppe and Jinghua Groppe. 2021. Optimizing Transaction Schedules on Universal Quantum Computers via Code Generation for Grover's Search Algorithm. In *25th Int. Database Engineering & Applications Symposium* (Montreal, QC, Canada) *(IDEAS '21)*. Association for Computing Machinery, New York, NY, USA, 149–156. https://doi.org/10.1145/3472163.3472164

[15] Lov K. Grover. 1996. A Fast Quantum Mechanical Algorithm for Database Search.. In *STOC*, Gary L. Miller (Ed.). ACM, Philadelphia, 212–219. http://dblp.uni-trier.de/db/conf/stoc/stoc1996.html#Grover96

[16] Le Gruenwald, Tobias Winker, Umut Çalıkyılmaz, Jinghua Groppe, and Sven Groppe. 2023. Index Tuning with Machine Learning on Quantum Computers for Large-Scale Database Applications. In *Proc. of QDSM@VLDB23*. https://ceur-ws.org/Vol-3462/QDSM5.pdf

[17] Stuart Hadfield. 2018. *Quantum Algorithms for Scientific Computing and Approximate Optimization*. Ph. D. Dissertation. Columbia University. arXiv:1805.03265 [quant-ph]

[18] Stuart Hadfield, Zhihui Wang, Bryan O'Gorman, Eleanor Rieffel, Davide Venturelli, and Rupak Biswas. 2019. From the Quantum Approximate Optimization Algorithm to a Quantum Alternating Operator Ansatz. *Algorithms* 12, 2 (Feb. 2019), 34. https://doi.org/10.3390/a12020034

[19] Torsten Hoefler, Thomas Häner, and Matthias Troyer. 2023. Disentangling Hype from Practicality: On Realistically Achieving Quantum Advantage. *Commun. ACM* 66, 5 (2023), 6 pages. https://doi.org/10.1145/3571725

[20] Toshinari Itoko and Takashi Imamichi. 2020. Scheduling of Operations in Quantum Compiler. In *2020 IEEE International Conference on Quantum Computing and Engineering*. IEEE Computer Society, Los Alamitos, CA, USA, 337–344. https://doi.org/10.1109/QCE49297.2020.00049

[21] Ahmed Kouider, Ammar Oulamara, Adlane Baaziz, and HACENE AITHADDADENE. 2022. Scheduling preemptive jobs on parallel machines with a conflict graph : A graph multi-coloring approach. *International Journal of Mathematics in Operational Research* 1 (01 2022), 1. https://doi.org/10.1504/IJMOR.2022.10049094

[22] Karim Lakhani, David Garvin, and Eric Lonstein. 2010. TopCoder (A): Developing Software through Crowdsourcing. *Harvard Business School General Management* Unit Case No. 610-032 (January 2010).

[23] Frank Leymann. 2019. Towards a Pattern Language for Quantum Algorithms. In *Proc. First International Workshop on Quantum Technology and Optimization Problems*. Springer International Publishing, Cham, Germany, 218–230. https://doi.org/10.1007/978-3-030-14082-3_19

[24] Elisabeth Lobe. 2023. quark: QUantum Application Reformulation Kernel. (2023). https://doi.org/10.18420/INF2023_123

[25] Andrew Lucas. 2014. Ising formulations of many NP problems. *Frontiers in Physics* 2 (02 2014), 5. https://doi.org/10.3389/fphy.2014.00005

[26] Mohammed Mahafzah. 2013. An Efficient Graph-Coloring Algorithm for Processor Allocation. *International Journal of Information Technology and Computer Science* 5 (06 2013), 43–48. https://doi.org/10.5815/ijitcs.2013.07.05

[27] Wolfgang Mauerer and Stefanie Scherzinger. 2022. 1-2-3 Reproducibility for Quantum Software Experiments. In *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE Computer Society, Los Alamitos, CA, USA, 1247–1248. https://doi.org/10.1109/SANER53432.2022.00148

[28] Nitin Nayak, Jan Rehfeld, Tobias Winker, Benjamin Warnke, Umut Çalikyilmaz, and Sven Groppe. 2023. Constructing Optimal Bushy Join Trees by Solving QUBO Problems on Quantum Hardware and Simulators. In *Proc. of BiDEDE@SIGMOD23*. Association for Computing Machinery, New York, NY, USA, Article 7, 7 pages. https://doi.org/10.1145/3579142.3594298

[29] Nikiforos Paraskevopoulos, Fabio Sebastiano, Carmen G. Almudever, and Sebastian Feld. 2023. SpinQ: Compilation Strategies for Scalable Spin-Qubit Architectures. *ACM Transactions on Quantum Computing* 5, 1, Article 4 (dec 2023), 36 pages. https://doi.org/10.1145/3624484

[30] John Preskill. 2018. Quantum Computing in the NISQ era and beyond. *Quantum* 2 (Aug. 2018), 79. https://doi.org/10.22331/q-2018-08-06-79

[31] Nils Quetschlich, Lukas Burgholzer, and Robert Wille. 2022. Compiler Optimization for Quantum Computing Using Reinforcement Learning. *2023 60th ACM/IEEE Design Automation Conference (DAC)* (2022), 1–6. https://api.semanticscholar.org/CorpusID:254535805

[32] Hila Safi, Karen Wintersperger, and Wolfgang Mauerer. 2023. Influence of HW-SW-Co-Design on Quantum Computing Scalability. In *2023 IEEE International Conference on Quantum Software (QSW)*. IEEE Computer Society, Los Alamitos, CA, USA, 104–115. https://doi.org/10.1109/QSW59989.2023.00022

[33] Lukas Schmidbauer, Karen Wintersperger, Elisabeth Lobe, and Wolfgang Mauerer. 2024. Polynomial Reduction Methods and their Impact on QAOA Circuits. In *IEEE International Conference on Quantum Software (QSW)*.

[34] Manuel Schönberger, Stefanie Scherzinger, and Wolfgang Mauerer. 2023. Ready to Leap (by Co-Design)? Join Order Optimisation on Quantum Hardware. *Proc. ACM Manag. Data* 1, 1, Article 92, 27 pages. https://doi.org/10.1145/3588946

[35] Manuel Schönberger, Immanuel Trummer, and Wolfgang Mauerer. 2023. Quantum-Inspired Digital Annealing for Join Ordering. *Proc. VLDB Endow.* 17, 3 (nov 2023), 511–524. https://doi.org/10.14778/3632093.3632112

[36] Manuel Schönberger, Immanuel Trummer, and Wolfgang Mauerer. 2023. Quantum Optimisation of General Join Trees. In *Proc. of QDSM@VLDB23*.

[37] Peter W. Shor. 1999. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Rev.* 41, 2 (1999), 303–332.

[38] Osvaldo Simeone. 2022. An Introduction to Quantum Machine Learning for Engineers. *Found. Trends Signal Process.* 16, 1–2 (jul 2022), 1–223. https://doi.org/10.1561/2000000118

[39] Unathi Skosana and Mark Tame. 2021. Demonstration of Shor's factoring algorithm for N = 21 on IBM quantum processors. *Scientific Reports* 11, 1 (Aug. 2021), 16599. https://doi.org/10.1038/s41598-021-95973-w

[40] Juexiao Su, Tianheng Tu, and Lei He. 2016. A quantum annealing approach for boolean satisfiability problem. In *Proceedings of the 53rd Annual Design Automation Conference* (Austin, Texas) *(DAC '16)*. Association for Computing Machinery, New York, NY, USA, Article 148, 6 pages. https://doi.org/10.1145/2897937.2897973

[41] Zsolt Tabi, Kareem H. El-Safty, Zsofia Kallus, Peter Haga, Tamas Kozsik, Adam Glos, and Zoltan Zimboras. 2020. Quantum Optimization for the Graph Coloring Problem with Space-Efficient Embedding. In *2020 IEEE International Conference on Quantum Computing and Engineering (QCE)*. IEEE Computer Society, Los Alamitos, CA, USA, 56–62. https://doi.org/10.1109/qce49297.2020.00018

[42] Kensuke Tamura, Tatsuhiko Shirai, Hosho Katsura, Shu Tanaka, and Nozomu Togawa. 2021. Performance Comparison of Typical Binary-Integer Encodings in an Ising Machine. *IEEE Access* 9 (2021), 81032–81039. https://doi.org/10.1109/ACCESS.2021.3081685

[43] Satish Thadani, Seema Bagora, and Anand Sharma. 2022. Applications of graph coloring in various fields. *Materials Today: Proceedings* 66 (2022), 3498–3501. https://doi.org/10.1016/j.matpr.2022.06.392

[44] Topcoder Contributer. 2019. Tutorial 1 - Sudoku | Topcoder Quantum Computing Challenge Series. Retrieved 2023-01-31 from https://tc3-japan.github.io/DA_tutorial/tutorial-1-sudoku.html

[45] Immanuel Trummer and Christoph Koch. 2016. Multiple Query Optimization on the D-Wave 2X Adiabatic Quantum Computer. *Proc. VLDB Endow.* 9, 9 (2016), 12 pages. https://doi.org/10.14778/2947618.2947621

[46] Manuela Weigold, Johanna Barzen, Frank Leymann, and Marie Salm. 2022. Data Encoding Patterns for Quantum Computing. In *Proceedings of the 27th Conference on Pattern Languages of Programs* (Virtual Event) *(PLoP '20)*. The Hillside Group, USA, Article 2, 11 pages.

[47] David P. Williamson and David B. Shmoys. 2011. *The Design of Approximation Algorithms*. Cambridge University Press, Cambridge, England. I–XI, 1–504 pages.

[48] Tobias Winker, Umut Çalikyilmaz, Le Gruenwald, and Sven Groppe. 2023. Quantum Machine Learning for Join Order Optimization using Variational Quantum Circuits. In *Proc. International Workshop on Big Data in Emergent Distributed Environments* (Seattle, WA, USA) *(BiDEDE '23)*. ACM, New York, NY, USA. https://doi.org/10.1145/3579142.3594299

[49] Tobias Winker, Sven Groppe, Valter Uotila, Zhengtong Yan, Jiaheng Lu, Maja Franz, and Wolfgang Mauerer. 2023. Quantum Machine Learning: Foundation, New Techniques, and Opportunities for Database Research. In *Companion of the 2023 Int. Conf. on Management of Data*. Association for Computing Machinery, New York, NY, USA, 45–52. https://doi.org/10.1145/3555041.3589404

[50] Karen Wintersperger, Hila Safi, and Wolfgang Mauerer. 2022. QPU-System Co-design for Quantum HPC Accelerators. In *Architecture of Computing Systems*, Martin Schulz, Carsten Trinitis, Nikela Papadopoulou, and Thilo Pionteck (Eds.). Springer International Publishing, Cham, 100–114.

[51] Biyuan Yao, Jianhua Yin, and Wei Wu. 2016. Deadlock Avoidance Based on Graph Theory. *International Journal of u- and e- Service, Science and Technology* 9 (02 2016), 353–362. https://doi.org/10.14257/ijunesst.2016.9.2.34

[52] Gongsheng Yuan, Yuxing Chen, Jiaheng Lu, Sai Wu, Zhiwei Ye, Ling Qian, and Gang Chen. 2024. Quantum Computing for Databases: Overview and Challenges. arXiv:2405.12511 [cs.DB]