Ready to Leap (by Co-Design)?
Join Order Optimisation on Quantum Hardware

Manuel Schönberger
Technical University of Applied Sciences Regensburg
Regensburg, Germany
manuel.schoenberger@othr.de

Stefanie Scherzinger
University of Passau
Passau, Germany
stefanie.scherzinger@uni-passau.de

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

ABSTRACT
The prospect of achieving computational speedups by exploiting quantum phenomena makes the use of quantum processing units (QPUs) attractive for many algorithmic database problems. Query optimisation, which concerns problems that typically need to explore large search spaces, seems like an ideal match for the known quantum algorithms. We present the first quantum implementation of join ordering, which is one of the most investigated and fundamental query optimisation problems, based on a reformulation to quadratic binary unconstrained optimisation problems.

We empirically characterise our method on two state-of-the-art QPUs for industrially relevant scenarios (neither was this achieved, to the best of our knowledge, by any of the prior research on using QPUs for databases or any other purpose).

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

SIGMOD ’23, June 18–23, 2023, Seattle, WA, USA
© 2023 Association for Computing Machinery.
ACM ISBN 978-1-4503-XXXX-X/18/06...
https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION
In recent years, quantum computing has attracted substantial attention in many fields of research, driven by the desire to benefit from quantum advantage in complex computations. While quantum computing has been studied for decades, the increase in interest aligns with the accelerating development of quantum computing hardware over the recent years, provided by vendors like IBM [36], Rigetti [66], or D-Wave [50], among many others. Moreover, cloud access to quantum systems has made quantum computing more accessible to researchers, enabling first experiments on real quantum processing units (QPU).

In contrast to CPUs, QPUs work with quantum bits, or qubits. Their mathematical state is exponentially larger than for classical bits, and they can realise phenomena like quantum superposition, quantum entanglement or quantum interference [58]. It is widely believed, given accepted complexity theoretic assumptions, that quantum systems offer increased computational power over classical systems [4, 7]. Speedups have been proven for multiple quantum algorithms [27, 70], and a seminal experiment [5] has demonstrated quantum advantage on real hardware, even if on an artificially constructed problem.

Quantum computers (QCs) are also believed to excel at optimisation problems that need to determine elements with specific properties in (exponentially large) search spaces, which is a commonly occurring problem in database systems, and particularly relevant for database query optimisation. However, QCs have so far seen only meagre adoption in DB research, and multi-query optimisation (MFO) [20, 74] is the only application in query optimisation that we are aware of, to the best of our knowledge.

We attribute this to the lack of practical utility of current prototype QPs, which are severely limited in various aspects. Given these limits, we cannot expect meaningful results for practical problems on current QPs, which are classified as so-called noisy intermediate scale quantum (NISQ) devices [62]. The goal of this paper is therefore not to demonstrate any practically relevant speedups for industrially relevant scenarios (neither was this achieved, to the best of our knowledge, by any of the prior research on using QPUs for databases or any other purpose).
One common assumption in previous work is that the performance of quantum approaches automatically improves with future universal QPUs. Unfortunately, this assumption is not a given [8, 62]: Firstly, universal large-scale, fully error-corrected QPUs are not expected within at least the next decade [34]. Secondly, as we show in this paper, the feasibility of employing QC is closely tied to an alignment of problem and QPU architecture. Even future universal QPUs will not automatically exhibit the required properties to achieve optimal quantum speedups for a specific problem, and will not automatically scale to the required problem dimensions without further ado.

Consequently, we quantitatively argue that one of the most pressing prerequisites to achieving near- and mid-term quantum advantage is the custom co-design of QPUs for specific problems. We outline a concrete path towards near-term DB-QPU utility that goes much beyond mere problem mappings. Rather than waiting on the sidelines, we actively aid QPU development, as we identify problem formulations with careful consideration of QPU properties, and derive design criteria that shape future QPUs as special purpose devices tailored to database applications.

We are the first to follow this QPU co-design principle for database problems. Specifically, we investigate the aptitude of quantum computing for the classic join ordering (JO) problem, which is one of the most extensively researched and fundamental problems in the field of query optimisation [25, 31, 43, 45, 52–54, 56, 57, 72, 75, 81], yet so far not addressed by prior quantum computing research. Since it is not possible to simply deploy existing classical algorithms on QPs, we reformulate JO into a mathematical description that is not possible to encode JO problems. In Sec. 6, we use insights gained from our experimental analysis for QPU co-design, and discuss tailored QPU improvements that greatly enhance near-term DB-QPU utility. We present related work in Sec. 7, and conclude in Sec. 8.

### 2 Quantum Foundations

Quantum computing is a relatively new computational paradigm, and prototypical hardware has only recently become available. It is impossible to provide a complete introduction to the field here. For a compact refresher focused on the quantum techniques used in this paper, we provide optional supplementary material. This section summarises the essential knowledge for our quantum approach. Detailed reviews on quantum computing (e.g., Refs. [8, 58]), annealing (e.g., Refs [2, 50]), and the utilised algorithms like QAOA [6, 21, 22, 29, 30, 32] are available in the literature.

**Quantum Algorithms.** Today, the biggest hurdle for using quantum computing is given by its incompatibility with classical algorithms. To solve JO on QPUs, we have to rely on quantum algorithms. Since quantum computation is inherently probabilistic, typical quantum algorithms aim to produce an optimal or near-optimal result with high probability. To obtain a statistical distribution, algorithms are executed tens to thousands of times, and an execution is referred to as shot.

Quantum algorithms require specific problem encodings. A typical formulation is the *quadratic unconstrained binary optimisation* (QUBO) encoding [9, 46], which (1) only allows quadratic, or pairwise, interactions between variables, (2) is *unconstrained* (i.e., does not allow for formulating explicit constraints), (3) only uses *binary* variables, and (4) encodes *optimisation* problems. Physically, we may interpret a QUBO problem as an *energy formula*, where we seek to determine the variable configuration corresponding to the minimum energy that encodes an optimal solution. Mathematically, QUBO problems are based on the multivariate polynomial

\[
f(\bar{x}) = \sum_i c_i x_i + \sum_{i \neq j} c_{ij} x_i x_j,
\]

where \(x_i \in \{0, 1\}\) are variables, and \(c_i, c_{ij} \in \mathbb{R}\) coefficients (it holds that \(c_{ij} = c_{ji}\)). Finding a JO-QUBO encoding, which we achieve in Sec. 3, allows us to solve JO with quantum algorithms.

The experimental analysis in Sec. 4 considers two algorithms that allow us to solve JO on two fundamentally different QPU architectures: (1) We run the *quantum approximate optimisation algorithm*
(QAOA) [21], which is an iteration-based, hybrid quantum-classical, algorithm to solve JO on gate-based QPUs. (2) We solve JO on quantum annealers [50], which allows us to draw a comparison of our JO approach to prior results for MQO [74] on this class of machines. While both problems are related to query optimisation, JO and MQO are structurally different, and known MQO to QUBO transformations do not apply for JO.

An important QAOA parameter is the chosen number of gate repetitions, typically denoted $p$. As shown by Farhi et al. [21], the approximation quality improves as $p$ increases. However, even for $p = 1$, QAOA shows promising experimental results for some problems [21, 22], and it is known that an efficient classical algorithm for sampling the output distribution of QAOA with $p = 1$ would imply a collapse of the polynomial hierarchy [23], which is seen as a clear indicator of possible quantum advantages.

**Gate-based QPUs.** For DB-QPU co-design in Sec. 6, we are mainly concerned with solving JO with QAOA on gate-based QPUs, as they allow for more design flexibility. As implied by the name, quantum gates perform operations on qubits in a quantum circuit, and thereby change their state to produce optimal or near-optimal solutions.

While in theory, QAOA solution quality increases for larger $p$, the required additional gates are detrimental on contemporary QPUs: A fundamental property of a quantum circuit is its depth (i.e., the longest sequence of operators in a circuit). Deeper circuits lead to a longer execution time, which is problematic: Unlike classical CPUs, QPU execution time is indirectly limited by the coherence time [58] of the hardware, which is, roughly speaking, the time during which quantum properties can be upheld. Increasing circuit depth increases the probability of decoherence errors because of quantum information loss due to environment interaction [65].

Circuit depth is therefore one of the most crucial metrics that we use to evaluate quantum computing feasibility, and is moreover substantially impacted by the properties of the QUBO encoding. For instance, to run the algorithm, the quantum circuit encoding the logical QUBO needs to be embedded, or transpiled, onto the physically realised QPU architecture, where connections between qubits are scarce. If the encoding requires connections between non-adjacent qubits, missing connections are established by inserting additional gates, thereby increasing depth. This interplay between problem encoding and QPU architecture remains characteristic for quantum computing, and further motivates QPU-DB co-design. We carefully consider such constraints for encoding JO in Sec. 3.

**QPU Metrics Summary.** Evaluating approaches for QPUs requires mastering a set of new metrics. To evaluate performance on contemporary QPUs, and to analyse the potential achievable by co-designed quantum systems, we consider the following metrics:

1. The overall QPU time required to obtain an optimal or near-optimal result with high probability;
2. For gate-based QPUs, the depth of the QAOA circuit determines execution time, but is also essential w.r.t. other QPU limits;
3. The number of qubits required by the QUBO encoding that needs to be fit onto QPUs.

### 3 JOIN ORDERING ON QPUS

For the JO-QUBO encoding, it is crucial to find a formulation that is well adapted to the constraints of QPUs. Fig. 1 gives an overview on all steps of our problem reformulation, where we

(1) express JO as mixed integer linear programming (MILP) problem,
(2) carefully adjust the MILP formulation to form a binary integer linear programming (BILP) model,
(3) transform BILP into QUBO, suitable for QPU processing.

In QUBO form, we can embed JO on QPUs, using established heuristic tools, and run quantum algorithms to find solutions. Owing to the probabilistic nature of quantum computing, we need to run thousands of shots, generating batches of results. In a final postprocessing step, we filter out invalid solutions, and read out the join order from valid ones.

#### 3.1 Formal Model

A JO problem is given by a set of relations $R_0, \ldots, R_n$, where $n_i$ denotes the cardinality of relation $R_i$, and a query graph, where each node represents a relation. An edge between relations $R_i$ and $R_j$ is labeled by a predicate $p_{ij}$ with selectivity $s_{ij}$, where $0 < s_{ij} \leq 1$ [16, 53]. Since our JO-QUBO transformation entails the JO-MILP formulation by Trummer and Koch [75] for classical solvers (but is unrelated to their work on MQO on QPUs), our approach follows their classification for JO: The approach seeks optimal left-deep join trees and allows cross products, without restrictions on the query graph (known to be NP-complete [16]).

We consider the classic cost function $C_{\text{out}}$ [16], which is given by

$$C_{\text{out}}(n_i, n_j) := n_i n_j s_{ij}.\]$$

Following Cluet and Moerkotte [16] to find the optimal join order for a sequence of $n$ relations $s_1, \ldots, s_n$, the cost function becomes

$$C(s) := \sum_{i=2}^{n} C_{\text{out}}(|s_1\ldots s_{i-1}|, |s_i|), \quad (2)$$

where $|s_1\ldots s_{i-1}|$ denotes the cardinality of the intermediate join result after joining $s_1, \ldots, s_{i-1}$.

**Model Extensions.** As we focus on near-term QPU-DB utility, we seek to keep qubit requirements as small as possible, to decrease the load on the limited capacity of contemporary QPUs. Therefore, we restrict our approach to the most basic MILP elements proposed in Ref. [75]. The problem remains NP-complete for this model.

Multiple extensions to the model are discussed in Ref. [75]. However, these require additional variables, which translate to higher qubit requirements. While we refrain from incorporating these for now, they can be freely added in the future alongside QPU advances, to cover even more complex scenarios. For instance, it is possible to model more sophisticated cost functions for a variety of operators, such as hash join and sort-merge join. Further extensions allow modelling correlations between predicates, which we, for now, assume to be uncorrelated. Finally, an extension to bushy trees may be possible. However, such an extension has yet to be found, and would likely yield a substantial qubit overhead.

#### 3.2 MILP Model and Pruning

In the first step, we encode JO as MILP. The original MILP model in Ref. [75] introduces additional variables to improve comprehensibility, relying on redundant variable elimination by the solver. Since
additional variables non-linearly translate to additional qubits, we need to reduce the model up-front to retain feasibility on restricted hardware. Modern MILP solvers can, to some extent, detect and prune redundancies [1]. However, as model size is crucial on QPUs, we manually ensure their removal. Additionally, precise knowledge of the influence of variables and constraints is important for the formal analysis in Sec. 5.

3.2.1 Overview. The JO-MILP model contains many variable types and constraints, and may be hard to follow. We nonetheless discuss it, to lay the foundation for subsequent sections. To aid the reader, Table 1 gives an overview of all MILP modelling steps. A running example moreover illustrates each modelling step in detail.

In essence, solving a MILP problem entails determining a value assignment for variables of integer or continuous domains, such that a given linear objective function is optimised [17]. For valid solutions, the variables need to satisfy a given set of linear constraints. For JO, MILP constraints are set to enforce a valid configuration of variables that represent various join ordering elements (e.g., join tree leaves, intermediate results and predicates).

Table 1 depicts the four modelling steps: (1) Enforce an unambiguous assignment of join tree leaves, and thereby a valid join order. To evaluate the join order, all subsequent steps contribute towards calculating the join costs. (2) Determine the outer operands, or intermediate results, for all joins. (3) Restrict the use of predicates if relations are missing from join operands. (4) Based on the intermediate results and predicate applicability, approximate the join cardinalities, or costs. The MILP objective function is set to minimise the approximated costs, to find the optimal join order.

Approximation is required since directly encoding the cost function into the MILP objective function is not possible due to the required product operations in \( C_{\text{opt}} \), which cannot be represented in the linear objective function with linear constraints. To circumvent this issue, Trummer and Koch [75] propose to use logarithmic cardinalities, as \( \log(\prod_i a_i) = \sum_i \log a_i \). Cardinalities are approximated via an arbitrary number of threshold variables. For quantum formulations, this results in a trade-off between better approximation and more qubits that requires great care.

3.2.2 Modelling Valid Join Orders. We follow approach and naming conventions of Ref. [75]. For each of the \( T \) relations and \( J \) joins, we distinguish between inner and outer operand in the left-deep join tree (outer operands are the result of preceding joins). The binary variables \( t_{ii0} \) (Table In Inner join order) and \( t_{ioj} \) (Table In Outer join order) indicate if relation \( i \) with \( 0 \leq t \leq T - 1 \), \( t \in \mathbb{N}_0 \) is part of the inner or outer operand of join \( j \) with \( 0 \leq j \leq J - 1 \), \( j \in \mathbb{N}_0 \). We add \( 2TJ \) such variables to the model (while only \( t_{ioj} \) variables for \( j = 0 \) are relevant for this step, we already add \( t_{ioj} \) variables for the remaining joins \( 1 \leq j \leq J - 1 \), for the next modelling step). To enforce solution validity, constraints (the latter added for each join \( j \))

\[
\sum_j t_{ioj} = 1, \quad \sum_t t_{ii0} = 1,
\]

ensure that each leaf of the join tree corresponds to exactly one relation, which enforces a valid join order.

For instance, for the outer operand of join 0, either \( t_{io0} \), \( t_{io0} \) or \( t_{ioj} \) must be set to 1, whereas the two remaining variables must equal 0, since exactly one relation must represent a join tree leaf. The same holds for the remaining leaf variables \( t_{ii1} \) and \( t_{ii1} \) for each relation \( t \). For this running example, we henceforth assume the join order \( R \bowtie S \bowtie T \), corresponding to the set of active (i.e., with value 1) variables \( \{t_{io0}, t_{ii0}, t_{ii1}\} \).

3.2.3 Modelling Intermediate Join Operands. For each join \( j > 0 \) and relation \( t \), the constraint

\[
t_{ioj} = t_{ii(j-1)} + t_{io(j-1)}
\]

ensures that a relation will be part of the outer operands of all subsequent joins once it is initially included in a join. Additional constraints ensure that the same relation cannot be part of both operands of a join. For all except the final join, these constraints are redundantly accounted for by the constraints in Eq. (4). It suffices to include such constraints for the final join and for each relation \( t \):

\[
t_{io(t-1)} + t_{ii(t-1)} \leq 1.
\]

Example 3.2. (cont’d) Consider again our example for the join order \( R \bowtie S \bowtie T \), represented by the active variables \( \{t_{io0}, t_{ii0}, t_{ii1}\} \). The newly added constraints ensure \( t_{io1} = 1 \), since \( t_{io1} = t_{ii0} + t_{io0} = 0 + 1 = 1 \). The same holds for \( t_{io1} \). As such, the outer operand for join 1 corresponds to the intermediate result after \( R \bowtie S \). We add the newly activated variables to our set of active variables: \( \{t_{io0}, t_{ii0}, t_{ii1}, t_{io1}, t_{io2}\} \).
3.2.4 Modelling Predicates. Consider binary predicates for joining two relations. For each predicate $p$ with $0 \leq p \leq P - 1, p \in \mathbb{N}_0$, where $P$ denotes the overall number of predicates, and for each join $j > 0$, we introduce a variable $pao_{pj}$ (Predicate Applicable in Outer Join). It indicates whether predicate $p$ can be evaluated for the outer operand of join $j$, requiring both associated relations as part of the outer operand of join $j$. For predicate $p$ and join $j > 0$, this is enforced by the constraints

$$pao_{pj} \leq tio_{T1(p)j}, \quad pao_{pj} \leq tio_{T2(p)j}, \quad (6)$$

where $T1(p)$ and $T2(p)$ denote the first and second relation for predicate $p$. The original model includes variables $pao_{pj}$, which we prune, since the outer operand of the first join contains only a single relation. In total, we add $P(J - 1)$ many $pao_{pj}$ variables.

Example 3.3. (cont’d) So far, $(tio_{00}, tio_{01}, tio_{10}, tio_{11}, tio_{S1})$ contains our active variables. We now consider the inclusion of join predicate $pRS$ for relations $R$ and $S$. One additional variable $pao_{01}$ is required to denote whether $pRS$ (indexed by 0) can be applied for join 1. The pruned model omits the superfluous variable $pao_{00}$ for join 0. Two constraints $pao_{01} \leq tio_{R1}$ and $pao_{01} \leq tio_{S1}$ enforce that $pao_{01}$ may only be set to one if both $R$ and $S$ are included in the outer operand of join 1. This is the case in our running example, as $pao_{01} \leq tio_{R1} = tio_{S1} = 1$, allowing $pao_{01} = 1$. The predicate may therefore be applied for the cardinality calculation. Similarly, we may add a predicate $pST$ or $pST$ for the relation $T$. If no predicate is provided, this necessitates a cross product for $T$. Adding the newly activated variable, our set of active variables becomes $(tio_{R0}, tio_{S0}, tio_{R1}, tio_{R1}, tio_{S1}, pao_{01})$.

3.2.5 Cost Function and Cardinality Approximation. Finally, based on the prior modelling steps for intermediate operands and predicates, we can approximate the associated costs for the join order. Estimating intermediate cardinalities can be encoded as a MILP problem, based on a logarithmic representation [75]. $c_j$ denotes the logarithmic cardinality for the outer operand of join $j$ by

$$c_j = \sum_t \log(Card(t)) \cdot tio_{tij} + \sum_p \log(\text{Sel}(p)) \cdot pao_{pj}, \quad (7)$$

where $Card(t) \geq 1$ is the cardinality of relation $t$, and $0 < Sel(p) \leq 1$ is the selectivity of predicate $p$. The cardinalities for each outer join operand are approximated using $R$ threshold values. For each threshold value $r$ with $0 \leq r \leq R - 1, r \in \mathbb{N}_0$ and each join $j$, a variable $cto_{tij}$ (Cardinality Threshold reached by Outer operand) is added to indicate if the intermediate logarithmic cardinality for the outer operand of join $j$ exceeds the threshold value $r$. If $cto_{tij} = 1$, the threshold value is added to the objective function

$$\min \sum_{r=1}^{R-1} \sum_{j=1}^{J} cto_{tij} \theta_r, \quad (6)$$

where $\theta_r$ denotes the $r$-th threshold value.

Example 3.4. (cont’d) We conclude our example by approximating the join costs, which, for three relations, merely consist of the cardinality resulting from the first join. We assume input cardinalities as $Card(R) = Card(S) = Card(T) = 100$ and $Sel(pRS) = 0.1$. For this simple scenario, optimal join orders are clearly given by $(R \bowtie S) \bowtie T$ and $(S \bowtie R) \bowtie T$, corresponding to the join costs $Card(R) \cdot Card(S) \cdot Sel(pRS) = 1,000$. However, for the MILP approach, we must approximate these costs using threshold values, which we assume to be $\theta_0 = 100$ and $\theta_1 = 1,000$. We add variables $cto_{0}$ and $cto_{1}$ for each threshold and for join 1, which has the intermediate result $R \bowtie S$ as the outer operand. As we use the...
cost function outlined in Eq. (2) and therefore only consider in-

termediate results, we do not add variables for join 0. For both vari-

bles, we add a constraint as given in Eq. (8). For \( c_j \), we obtain

\[
c_j = \log(\text{Card}(R)) \cdot \delta_0 + \log(\text{Card}(S)) \cdot \delta_1 + \log(\text{Card}(T)) \cdot \delta_2 + \log(\text{Sel}(p_{GS})) \cdot \rho_{\text{opt}} = 2 \cdot 1 + 2 \cdot 1 + 2 \cdot 0 \cdot 1 = 3.
\]

Since \( \log(3) = \log(100) = 2 \cdot \log(100) = 2, \quad \delta_0 = 1 \) to satisfy the constraint. However, since \( 3 \leq \log(100) = 3 \), \( \delta_0 = 0 \) satisfies the constraint. Therefore, only \( \delta_0 = 0 \) is added to the costs, which is far from the true intermediate cardinality for \( R \Rightarrow S \). The approximation can be improved by adding more threshold variables. Evidently, the choice of threshold values greatly impacts the accuracy. This requires careful consideration for QPUs, where finding a balance between sufficient accuracy and qubit count is crucial.

Table 2 summarises savings in variables and constraints by prun-
ing the original MILP model of Ref. [75]. The importance of these

savings will be highlighted by the following sections, which de-
scribe how variables and constraints translate to qubit requirements and further QPU load, which decisively influences the feasibility
on NISQ machines.

3.3 BILP Formulation

To transform MILP to QUBO, as required by QPUs, we build on an
intermediate BILP\(^*\) step, since efficient transformations from BILP
to QUBO are known [47], at least for problems restricted to binary
variables and equality constraints. The pruned MILP model includes

inequality constraints that we turn into equality constraints by
adding additional variables [17], typically called slack variables.
For instance, converting the constraint in Eq. (8) by adding a slack
variable \( s_{rj} \) gives

\[
c_j = \log(\text{Card}(R)) \cdot \delta_0 + \log(\text{Card}(S)) \cdot \delta_1 + \log(\text{Card}(T)) \cdot \delta_2 + \log(\text{Sel}(p_{GS})) \cdot \rho_{\text{opt}} = 2 \cdot 1 + 2 \cdot 1 + 2 \cdot 0 \cdot 1 = 3. \text{ Since } \log(3) = \log(100) = 2, \quad \delta_0 = 1 \text{ to satisfy the constraint. However, since } 3 \leq \log(100) = 3, \quad \delta_0 = 0 \text{ satisfies the constraint. Therefore, only } \delta_0 = 0 \text{ is added to the costs, which is far from the true intermediate cardinality for } R \Rightarrow S. \text{ The approximation can be improved by adding more threshold variables. Evidently, the choice of threshold values greatly impacts the accuracy. This requires careful consideration for QPUs, where finding a balance between sufficient accuracy and qubit count is crucial.}

Table 2 summarises savings in variables and constraints by prun-
ing the original MILP model of Ref. [75]. The importance of these

savings will be highlighted by the following sections, which de-
scribe how variables and constraints translate to qubit requirements and further QPU load, which decisively influences the feasibility
on NISQ machines.

3.3 BILP Formulation

To transform MILP to QUBO, as required by QPUs, we build on an
intermediate BILP\(^*\) step, since efficient transformations from BILP
to QUBO are known [47], at least for problems restricted to binary
variables and equality constraints. The pruned MILP model includes

inequality constraints that we turn into equality constraints by
adding additional variables [17], typically called slack variables.
For instance, converting the constraint in Eq. (8) by adding a slack
variable \( s_{rj} \) gives

\[
c_j = c_{\text{to}} r_j \cdot \delta_0 r_j + s_{rj} = \log(\theta_j). \quad (9)
\]

However, a binary domain for \( s_{rj} \) is insufficient to ensure the trans-
formed equality is functionally equivalent to the original inequality.
Instead, continuous values are required for \( s_{rj} \), which violates the
restriction to binary variables. We therefore approximate \( s_{rj} \) by
discretising with multiple binary variables, since an integer
bounded by \( C \) can be expressed using \( n = \lceil \log_2(C) \rceil + 1 \) binary
variables [13]. This gives \( s_{rj} \simeq \omega \cdot \sum_{i=1}^{N} 2^{i-1} b_i \), where \( \omega \) denotes the
discretisation precision, and \( b_i \in \{0,1\} \). This results in

\[
n = \lceil \log_2(C/\omega) \rceil + 1 \quad (10)
\]

binary variables, which, again, leads to a trade-off: More binary
variables for the discretisation (smaller \( \omega \)) lead to higher preci-
sion, which is costly given the limited number of qubits. However,
without any remaining inequality constraints, the BILP problem
can now be trivially cast as QUBO suitable for QPUs.

3.4 QUBO Formulation

**Encoding.** Unlike BILP, QUBO problems cannot include explicit

constraints to enforce validity. Instead, we need to ensure that a
solution with minimum value inerently corresponds to a valid

\[\text{subject to } A\vec{x} = \vec{b}, \quad x_i \in \{0,1\}, \quad i = 1,\ldots,m\]\n
solution. Lukas [47] provides a conversion that turns BILP problems

with \( N \) variables into QUBOs with \( N \) qubits of the form

\[
H = H_A + H_B = A \sum_{j=1}^{m} \left( b_j - \sum_{i=1}^{N} S_{ji} x_i \right)^2 + B \sum_{i=1}^{N} c_i x_i, \quad (11)
\]

where \( H_A \) encodes the BILP constraints \( Sx = b \). The inner quadratic

term evaluates to 0 iff no constraint is violated. Invalid solutions

are penalised by \( A \), and cannot correspond to the minimum value.

Through the discretisation of continuous variables, \( b_j - \sum_{i=1}^{N} S_{ji} x_i \)

may only be close, but not equal to 0 even for valid solutions. To
cumvent this problem, we round the coefficients \( S_{ji} \) according to
the discretisation precision \( \omega \). Term \( H_B \) encodes the cost.

Weights \( A \) and \( B \) in Eq. (11) must be suitably assigned. Since we

prioritise valid and non-optimal solutions over optimal but invalid
ones, \( A \gg B \) needs to hold. The weights cannot be set to arbitrarily
large values, as high penalty weights are known to cause issues like
slowdowns [59]. We therefore choose the smallest possible weights
such that violating a single constraint by the smallest possible
amount already leads to a sufficiently large penalty such that the
solution cannot correspond to the minimum energy.

To determine this smallest possible violation, first consider con-
trasts with only binary variables. The minimum violation is 1 in
this case—for instance, in constraint \( 1 - x_1 + x_2 = 0 \), with binary
variables, the configuration \( x_1 = x_2 = 1 \) leads to a violation by 1, and
contributes penalty \( A \cdot 1^2 = A \). In contrast, constraint \( c-1.1 = 0 \)
with continuous variable \( c \) discretised at precision \( \omega = 0.1 \) delivers
a minimal violation for \( c \approx 1.2 \) or \( c \approx 1.0 \), contributing a penalty of
\( A(0.1)^2 \approx A(\omega)^2 \). Therefore, the smallest violation is given by

\( \omega \) and hence, \( A = C/\omega^2 + \epsilon \) for \( B = 1 \), where \( \epsilon \) is some small value and \( C = \sum_{i=1}^{n} c_i \). Violating a constraint in \( H_A \) to save costs that would
otherwise be added in \( H_B \) is thereby discouraged, as it will always
lead to the same or even larger costs.

**Quadratic Contributions.** So far, we only addressed how to limit
the number of qubits by reducing the number of variables in the
problem model. Equally important properties like circuit depth for
gate-based QPUs are strongly influenced by the required pairwise,
or quadratic, qubit interactions. Consequently, understanding the

genesis of quadratic contributions to Eq. (11) is required.

Only \( H_A \), encoding constraints, can entail quadratic interac-
ctions. A quadratic contribution arises for each pair of variables
that appears in at least one constraint. Of all constraints derived
in Sec. 3.2, cardinality approximation in Eq. (8)—required for every

join and every threshold value—contains the most variables (other

constraints contain at most three binary variables, including slack

variables). For Eq. (8), quadratic contributions arise for all variables

\( \delta_0 j, \delta_1 j \) and \( \delta_2 j \) associated with join \( j \). Pairs between these

variables and variable \( c_{\text{to}} j \), as well as all binary variables needed
to express the slack variable \( s_{rj} \), are required.

**Choosing Approximation Precision.** As discussed above, a higher
precison via increasing the number of threshold values comes at the
expense of higher qubit requirements. It also leads to an increasing

amount of quadratic contributions in Eq. (11), as it impacts the
number of required cardinality approximation constraints. The
discretisation precision of continuous slack variables has a similarly
large impact, as it influences the amount of binary slack variables
in each of these constraints.

QPU limitations should be considered when choosing suitable
threshold values. For instance, QPU load can be reduced by setting
integer logarithmic thresholds, since these do not require value dis-
cretisation, and not only save qubits for encoding larger problems,
but also reduce quadratic QUBO contributions. In general, rather
than applying an abundance of thresholds, high QPU utility can be
best achieved by choosing few, yet carefully selected values, which
may be based on available empirical knowledge on intermediate
cardinality sizes, to cover a broad range of common scenarios.

The strict necessity to carefully choose approximation precision
is uncustomed from classical approaches, yet characteristic in a
quantum context. Our model allows us to freely tune both precisions
(thresholds and discretisation precision). This allows us to influence
the amount of quadratic contributions, and their impact on the
QAOA circuit depth.

3.5 Postprocessing
The final QUBO model in Eq. (11) can be used to solve JO using both,
gate-based QAOA, and quantum annealing.3 From either hardware,
we obtain a batch of possible QUBO variable assignments together
with the corresponding value of Eq. (11) that indicates solution
quality. Each result must be mapped back to the initial JO prob-
lem. Since QPUs can deliver invalid solutions owing to hardware
imperfections, we need to verify solutions in a postprocessing step.

Since non-optimal and invalid solutions are penalised, it is often
possible to filter by the magnitude of the final penalty value. Caused
by the multi-step reduction to QUBO, a large penalty indicates an
invalid solution to the BILP problem with at least one constraint
violation, but does not necessarily imply an invalid JO solution.

A solution is valid when an unambiguous, valid join tree can be
derived from the assignment of QUBO variables, even if some
constraints (e.g., those relevant for the calculation of intermediate
 cardinalities) are violated. Instead of judging a solution by its pen-
alty value, we consider the value assignments for all \( t_{ii} \) variables,
which indicate the relations selected as the inner join operands,
and verify whether each inner operand is uniquely represented by
exactly one input relation. The final relation, representing the outer
operand of the first join, is then given by process of elimination.

To judge solution quality, we calculate costs of the resulting join
trees, and determine the best join order among all valid solutions.

4 ASSESSING STATE-OF-THE-ART QPUS
Contemporary QPUs are not expected to solve practically relevant
instances of JO. However, despite the limitation to small-sized prob-
lems, experiments on actual hardware are beneficial for two reasons:
Firstly, they provide the only means to properly evaluate the poten-
tial and soundness of our approach, as resource requirements for
simulating larger QPUs to tackle practically sized problems scale
exponentially, quickly rendering simulations infeasible. Secondly,
they provide insights on what limitations should ideally be ad-
dressed in future QPUs to gain speedups over classical approaches.

Prior results for query optimisation on QPUs are based on ex-
periments on D-Wave quantum annealers [74]. Therefore, we first
analyse JO on D-Wave QPUs, and relate our findings for JO to these
prior results for MQO. Like [74], we observe speedups for small
problem dimensions, but point out substantial scalability issues.
Comparing JO and MQO results provides evidence for the potential
of co-designing QPUs for DB problems to ensure scalability.

We then commence to analyse JO on current gate-based QPUs,
since these are more suitable for co-design. While these QPUs are
similarly limited, and offer no practical utility, we study the impact
of specific problem parameters on quantum computing feasibility,
and derive insights for co-designing future QPUs for JO.

4.1 Experimental Setup
For quantum annealing experiments, we resort to the D-Wave Ad-

vantage [50] system (around 5000 qubits, recently improved with a
 performance update, Pegasus topology).

For gate-based QAOA, we consider the IBM Q Auckland (27
qubits, Falcon r5.11 topology) QPU. At the time of writing, IBM Q
Washington (127 qubits, Eagle r1 topology) is the largest IBM Q
system in terms of qubits, but with marked disadvantages in coher-
ence time. Our results for circuit embeddings confirm that these
disadvantages outweigh the benefits. We restrict QPU performance
analysis to the smaller, more stable Auckland QPU.

Algorithmic Setup. Our approach, implemented in Python, pre-

dares QUBO formulations for a given JO problem, and handles
interaction with the QPUs. It relies on library gurobi.py [28] for
formulating MILP and BILP problems. Packages docplex [35] (for
IBM Q) and qubovert [64] (for D-Wave), are used to formulate
QUBO representations, which are then processed and passed to the
QPU by Qiskit [38] (IBM) and D-Wave Ocean [18].

For our experiments on the D-Wave Advantage system, we de-
termine suitable embeddings using the heuristic minorminer tool
provided in the D-Wave Ocean library [19]. The QPU runtime, or
annealing time, is a parameter for D-Wave QPUs. We conduct ex-
periments for varying annealing times \( t \), to study their impact on
the solution quality. Since the number of allowed shots decreases
with increasing annealing time per shot, we perform 10,000 shots
for \( t < 10 \mu s \) and 900 shots for \( t \geq 10 \mu s \). Other parameters include
the chain strength, which groups physical qubits into a chain repres-
enting a logical qubit (for detailed information, see Ref. [40]). We
experimentally determine suitable chain strengths, depending on
JO problem sizes. The exact values can be found in our reproduction
package. Remaining parameters are set to their default values.

The Qiskit QAOA library is used to generate quantum circuits
that can be embedded onto IBM Q using the Qiskit transpiler (opti-
misation level 1). We run QAOA with \( p = 1 \), where \( p \) denotes the
number of QAOA operator repetitions, since larger values for \( p \) lead
to circuit depths beyond machine capability. Classical optimisation
is performed with the Qiskit analytic quantum gradient descent
optimiser (AQGD). We sample 1,024 shots for each circuit executed
on the QPU. Remaining parameters are set to their defaults.

3Note that the general structure of a QAOA circuit does not depend on the objective
QUBO function [21]. Depicting the resulting quantum circuit for an encoded JO-QUBO
is infeasible as it consists of thousands of gates, and would provide no additional
insights. We refer interested readers to Ref. [21] and our quantum fundamentals
supplement, where the generic QAOA circuit structure is discussed.
Queries. Our goal is to find bounds on the dimension of JO problems for which QPUs determine viable solutions. QPU performance is influenced by a multitude of complex factors, both algorithmic and physical, and deteriorates quickly for increasingly sized problems. That is why we place particular emphasis on the generation of input data: On the one hand, we want to isolate effects of QPU imperfections; on the other hand, we need address scenarios with realistic characteristics.

Input data for the JO problem does not consist of SQL statements, but comprises (statistical) data derived from the relations and predicates involved in an SQL statement. Specifically, input data consists of a query graph (annotated with cardinalities), applicable join predicates with (estimates of) their selectivities, and the threshold values for approximating intermediate cardinalities.

Input data with realistic, representative workload characteristics can be created in two ways: (1) Starting from SQL statements, we can extract required statistical information, and then generate the input query graph. (2) We can directly generate input query graphs alongside a distribution of representative relation cardinalities, predicates and selectivities. The latter method has, for instance, been employed to evaluate the JO-MILP approach of Ref. [75].

Input for the JO problem is, in each case, given by a query graph. To avoid parsing SQL, statistical estimation, and other non-quantum tasks, we apply approach (2). We consider both, multiple generation methods (seminal and recent) for realistic query loads, and state-of-the-art JO algorithms executed on recent classical hardware.

We use (a) the query generation from Ref. [73], which creates queries based on the seminal method of Steinbrunn, Moerkotte and Kemper from VLDB’97 [72] for benchmarking JO approaches. Their implementation supports generating chain, star and cycle queries. Additionally, we (b) generate clique queries (i.e., queries with join predicates for each pair of relations) to compare our approach to recent classical JO results presented by Mancini et al. [48], and generate queries using the exact same method as in their paper. Since the methods considered in Ref. [48] do not allow for considering cross products, whereas our approach does, we base our comparison on clique graphs. All join pairs in this case are valid, which is equivalent to a cross-join scenario without the need for explicit cross join support in the base method. We find this to enable the fairest comparison.

For cases (a) and (b), we generate sets of 20 queries for different problem size scenarios, where we consider between two and five relations (we find below that larger queries exceed the limited capacity of our subject QPUs).

Additionally, we (c) generate input data with cardinalities, selectivities and threshold values hand-crafted that try to minimise the expected influence of imperfections of current QPUs on solution quality. As pointed out in Sec. 3, current QPUs require discretising continuous variables, at the expense of reducing QPU capacity for increasing precision. We restrict this set of input data to integer logarithmic cardinalities, predicate selectivities and threshold values, as they do not require value discretisation. Data and code for all methods are supplied in the reproduction package.

Limitations of larger IBM Q machines, as explained above, necessitate restricting experiments to 27 qubit (Auckland) machines. We can process basic queries that join at most three relations. Experiments do not reach practically relevant dimensions, but we provide qubit requirements for realistic problems on future QPUs in Sec. 6. Compared to IBM Q, the D-Wave Advantage system allows for embedding larger queries, although limits are also reached quickly, as shown below.

Even with the restriction to three relations, we can study JO problems with varying properties on IBM Q systems. For instance, we generate problems that consider different numbers of predicates: For a query joining three relations, this provides us with four scenarios in total, where the number of predicates ranges between zero and three. As a result, for queries with less than two join predicates, cross products are required. For the remaining two scenarios with two and three predicates, we generate a chain query and a cycle query. These scenarios translate into varying qubit requirements, from 18 qubits for zero predicates up to 27 qubits for three predicates. This allows us to judge the impact of increasing problem dimensions, even with the restriction to three relations. Similarly, instead of increasing the number of predicates, we can vary the precision for discretising continuous variables, and generate problems of different dimensions, ranging from 18 to 27 qubits. This allows us to compare specific parameter settings, to identify parameter-specific bottlenecks.

4.2 Experimental Results

We next discuss the experimental evaluation on state-of-the-art QPUs, focusing on (a) run-time, (b) result quality, and (c) maximal problem sizes (qubit count, circuit depth).

4.2.1 Join Ordering on D-Wave

Experimental Results. For QA, runtime is a parameter: If it is too small, the probability for a non-optimal or invalid solution increases; if it is too large, this is obviously counter to speedups. The overall QPU runtime essentially depends on the annealing time per shot $\Delta t$ and the number of shots $n$ required until an optimal solution is found; Tab. 3 shows observed experimental values. Since $n$ is a statistically distributed quantity, we compute the average value $\bar{n}$ to determine the average runtime $T = \bar{n} \cdot \Delta t$. As expected intuitively, increasing $\Delta t$ reduces $\bar{n}$. $T$ is minimal for a (short) annealing time of $\Delta t = 0.5\mu s$, which we use in the following experiments.

To evaluate the potential of our JO-QPU approach, we need to compare it against an established baseline. However, the criteria for fair QPU-CPU comparisons are subject of scientific debate [51]. Therefore, we can neither demonstrate universal speedups over all parameter ranges, nor show practical utility. At the same time, such comparisons are vital to demonstrate the potential of QPU advantages to the DB community.

Mancini et al. [48] have, at SIGMOD’22, presented a massively parallel, efficient JO algorithm that they compare to some of the fastest known state-of-the-art JO approaches (Postgres, DPCCP [54], DPE [31], DPSub [52], DPSize [52]—it is beyond the scope of this paper to discuss these classical algorithms in detail). To the best of our knowledge, this is the only reference in the literature with a sole focus on join-ordering that is (a) very recent, (b) addresses...
an input data scenario comparable to ours, (c) covers a broad selection of JO algorithms, and (d) gives quantitative optimisation time measurements. We compare our QPU results inferred for all input data scenarios to their corresponding classical results.

Fig. 2 compares our quantum results against their published classical numbers. Since our approach is inherently stochastic, we provide the runtime distribution as compared to the mean values reported by Mancini et al. [48] for state-of-the-art classical algorithms. The middle vertical line in the boxplots represent our observed median results.

To identify the impact of discretisation costs on QPUs, we compare solution time for queries with integer logarithmic values, which do not require any variable discretisation, against solution time for Steinbrunn and Mancini queries, which capture realistic scenarios, where we set the discretisation precision to two decimal positions, which we determined as the best choice (i.e., resulting in the highest solution quality) for the considered queries after preliminary testing.

For small problem dimensions, we observe pronounced speedups—encompassing multiple orders of magnitude!—over all classical algorithms provided in Ref. [48]. While not considering these results direct evidence, as we stress again, for the practical feasibility of our approach, we do see them as a promising indication.

These speedups are almost equally pronounced for both, integer logarithmic queries, and the more general Steinbrunn and Mancini queries, and the impact of discretisation costs seems negligible for the considered small-scale problems: QPUs can attain optimal solutions despite the higher QPU load due to discretisation. Neither do the specific query cardinalities and selectivities seem to exert much impact on QPU performance.

It therefore suffices to consider one set of input data in the following, and we choose integer logarithmic queries. This allows us to identify upper bounds independent of QPU discretisation costs, and also reduces the pecuniary impact of QPU access somewhat.

While we find the results for small-scale queries very promising, great care needs to be applied when interpreting the observations, and especially when extrapolating to larger instances. Table 3 lists the probabilities of reaching valid and optimal solutions for the more common chain, star and cycle graphs with up to five relations (minimal discretisation precision, one inequality threshold value). Solution quality is not much influenced by query graph topology, but declines quickly with an increasing number of relations (the impact of annealing times is minimal, which agrees with previous observations in the literature [44, 69]). In general, it suffices if one annealing shot delivers a (near-)optimal solution; this is the case for three and four relations. However, for five relations, almost none of the 900 shots include a valid, and none an optimal solution—dependent of query graph type and annealing time. This is likely attributable to the magnitude of perturbations in a fully utilised annealer [51].

Another impediment is that technical issues like communication between QPU and CPU must be taken into account. While these could be reduced to negligible levels, they can—depending on the actual setup—exceed the sampling time by orders of magnitude larger than the sampling time. This must be taken into account when evaluating indiscriminately optimistic claims on possible QPU speedups, as seen in the literature.

We conclude that while our results indicate substantial potential, solving JO problems with more than four relations is infeasible for current D-Wave QPUs. Unfortunately, simulating large, noisy quantum systems as would be required to provide an estimate of the performance of future QAs in the absence of real hardware is itself an NP-complete problem that is beyond reach [8] (if simulating such systems were feasible, there would be no need to build quantum annealers in the first place).

Table 3: Average fraction of valid and optimal solutions obtained in 1,000 annealing runs (DWave) over 20 JO experiments, depending on annealing time ($\Delta t$) and query graph.

<table>
<thead>
<tr>
<th>Query Graph</th>
<th>$\Delta t$ [µs]</th>
<th>3 Relations</th>
<th>4 Relations</th>
<th>5 Relations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chain</td>
<td>10</td>
<td>31.00%</td>
<td>8.69%</td>
<td>1.23%</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>37.01%</td>
<td>8.22%</td>
<td>1.53%</td>
</tr>
<tr>
<td></td>
<td>1000</td>
<td>41.25%</td>
<td>10.30%</td>
<td>1.78%</td>
</tr>
<tr>
<td>Star</td>
<td>10</td>
<td>-</td>
<td>2.16%</td>
<td>0.33%</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>-</td>
<td>2.01%</td>
<td>0.29%</td>
</tr>
<tr>
<td></td>
<td>1000</td>
<td>-</td>
<td>2.48%</td>
<td>0.43%</td>
</tr>
<tr>
<td>Cycle</td>
<td>10</td>
<td>27.52%</td>
<td>11.46%</td>
<td>3.32%</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>31.10%</td>
<td>14.36%</td>
<td>3.77%</td>
</tr>
<tr>
<td></td>
<td>1000</td>
<td>34.58%</td>
<td>16.89%</td>
<td>4.21%</td>
</tr>
</tbody>
</table>

Context. Let us set these results into context with the findings of the only previous study on using a quantum annealer for query optimisation. Trummer and Koch [74] also compare a quantum approach against classical benchmarks, but for a different problem (MQO instead of JO). They find advantages for larger problems which albeit far from reaching practical dimensions, surpass soluble problem sizes in our case.

A fundamental difference between these approaches explains this discrepancy: In contrast to using heuristic tools to determine
embeddings for JO, the MQO-QUBO is mapped to the physical hardware using a hand-crafted embedding. Most importantly, the fixed patterns of MQO agree well with the QPU connectivity structure.

It is not possible to enjoy these benefits for JO (and most other problems): It is only suitable for problems with predictable and fixed structures independent of the input, whereas pairwise qubit interactions in JO depend on the problem instance. Restricting problems to fixed structures also restricts generality and scalability. Also, the embedding problem is NP-complete [49, 82], and any manual approach is deemed to quickly become infeasible. However, the MQO results suggest a pronounced link between solution quality and problem alignment with quantum hardware.

Instead of trying to align different problems with unique requirements to QPUs with fixed architecture, we instead suggest to tailor QPUs to DB problems via DB-QPU co-design. Since gate-based QPUs offer more design flexibility [8] they are better suited to co-design than quantum annealers. Consequently, we next analyse the performance of our approach on gate-based IBM-Q QPUs, and then consider how co-design can improve their performance toward practical utility in Sec. 6.

4.2.2 Join Ordering on IBM Q. For IBM Q hardware, we analyse circuit depths for varying JO problems and QPUs in addition to the observables considered in the previous section. The depth of a circuit on a gate-based QPU crucially impacts feasibility; with increasing depth, quantum coherence is lost, and result quality degrades.

Figure 3: Circuit depths of various scenarios, IBM Q devices.

Circuit Depths. While the number of required qubits is a metric for problem size, the circuit depth can be seen to quantify the impact of specific problem parameters. These require varying interactions between qubits, and lead to different depths. Therefore, two problems with identical size (in qubits) may produce substantially different circuit depths, which can likewise substantially impact the solution feasibility on a QPU.

This scenario is depicted in Fig. 3, which shows the distribution of circuit depths (based on 20 transpilations) necessary to embed QAOA circuits on two IBM Q devices using the heuristic Qiskit transpil. The left hand side of Fig. 3 compares circuit depths for problems with three relations and one threshold value, which number of predicates and approximation precisions. By varying the discretisation precision (striped boxplots) from zero to three decimal places (using zero predicates), we arrive at problems that can be mapped to 18, 21, 24 and 27 qubits. By varying the amount of predicates (dotted boxplots) from zero to three (using a discretisation precision of 0 decimal positions), we arrive at the same qubit requirements. We see that the increase in median circuit depth, but also in variance is considerably more pronounced for increasing precision than for increasing amounts of predicates. Equally interesting, the transpilation heuristic alone contributes substantial variance in the 24 and 27 qubit cases. Increasing the discretisation precision therefore not only limits the feasible instance size (as few qubits are available for encoding relations or predicates), but also greatly impacts the circuit depth, thereby increasing the probability for gate errors and decoherence errors, and reducing result quality.

The results suggest that qubit connectivity offered by current QPUs is a major impediment to scalability, since a higher discretisation precision demands more qubit interactions. Our co-design simulations of improved QPU architectures in Sec. 6 confirm this hypothesis.

The right hand side of Fig. 3 compares embeddings of JO problems with an increasing number of predicates on IBM Q Auckland (27 qubits, Falcon r5.11 topology) and Washington (127 qubits, Eagle r1 topology) QPUs to analyse the impact of different qubit topologies on circuit depth. The increase is comparable to varying predicates on the left hand side, and we observe up to 70% dependence depending on the slight variations of the topology for one single vendor (interestingly, the larger connectivity graph in terms of qubits leads to higher circuit depths).

We need to put these results in context by considering coherence times and average gate time $g_{avg}$—the cumulative gate times form a lax upper bound for practical utility, because for longer computation times, random results are to be expected. At the time of running the experiment, the systems report coherence times of $T_1 = 151.13\mu s$, $T_2 = 138.72\mu s$ (Auckland), and $T_1 = 92.81\mu s$, $T_2 = 93.36\mu s$ (Washington). Average gate times are reported as $472.51\mu s$ (Auckland) and $550.41\mu s$ (Washington). Given these parameters, the approximate maximum depth $d$ of a circuit to not exceed $T_1$ or $T_2$ is given by $d = \frac{\min(T_1, T_2)}{g_{avg}}$. We thereby get $d_q = 293$ for the Auckland QPU and $d_q = 168$ for the Washington QPU. Even for scenarios with less discretisation precision, the maximum depth is quickly reached for Washington, where we therefore cannot expect meaningful results, outweighing its size benefits. This, again, shows that waiting for QPUs with higher qubit capacity does not guarantee more favourable system-global properties, and rather motivates us to better align problems and topologies. Even for Auckland, problems with a high precision exceed this depth. For experimentally analysing QPU performance, we therefore only analyse JO problems at the minimum precision, using the Auckland QPU.

QAOA Performance. Following the stability considerations

| Table 4: Solution quality for $n$ QAOA iterations (three relations). |
|-----------------|--------|--------|--------|--------|
| Predicates/Qubits | 0/18 | 1/21 | 2/24 | 3/27 |
| $n$ | 3% | 7% | 13% | 13% |
| $n=1$ | 20 | 3% | 4% | 4% |
| Opt. | 50 | 3% | 3% | 5% |

5IBM QPUs undergo periodic recalibration operations to optimise $T_1$ and $T_2$, so the values are not stable across different experiments.

6Quantum volume [55] has been suggested as balancing all of these various characteristics of QPUs, indicating a better balance between transpilations and experiments.
from above, we restrict our performance experiments to the 27-qubit Auckland QPU, which allows for working JO problems with three relations before the available qubit capacity is exhausted. Table 4 shows QAOA results for 1,024 shots, for queries with increasing predicate numbers, and provides the fraction of measurement shots that represent valid or optimal solutions.

For all considered problems, optimal solutions are determined by the QPU. Interestingly, for increasing problem dimensions, we do not observe a consistent decrease in the ratios of either valid or optimal solutions. Likewise, increasing the number of QAOA iterations has no consistent impact on solution quality. Due to current QPU limitations, it remains an open question how the solution quality scales for larger search spaces (i.e., more relations).

Finally, consider the time $t_i$ required to perform the actual circuit sampling for one QAOA iteration. For 0 predicates, $t_i = 77.9$ms. The time increases to $t_i = 113.7$ms for three predicates. Given that the average number of classical QAOA iterations steps (using the COBYLA optimiser) required until convergence is 28.5, it is obvious that this type of QPU is currently unable to compete with the classical approaches summarised in Fig. 2, and is not practically useful to solve JO problems. Limited amounts of qubits, limited connectivity, and lack of robustness against noise reduce capabilities and performance. These issues will eventually (likely in the very long term [42]) be addressed by error-corrected, perfect QPUs.

To progress towards earlier QPU utility, we pursue two strands: First, we derive an upper bound on the amount of required qubits for JO; this allows us to make suggestions for qubit capacities sufficient for realistic queries, yet also leaves room to address additional limitations like circuit depth. Using insights gained from our experimental analysis, we then address the question of which properties of QPUs should be improved to progress towards practical utility via co-design (recall that we have argued above why gate-based QPUs are a better candidate for co-design efforts than annealers).

## 5 FORMAL ANALYSIS

We now derive upper bounds on the amount of logical qubits based on the BILP model, where each variable corresponds to one qubit.

Slack variables, as required to handle inequalities, contribute substantially to the overall number of qubits. Before we can derive an upper bound for the number of binary slack variables, we need to bound the value of a continuous slack variable:

**Lemma 5.1.** The value for a continuous slack variable $s_{rj}$ is bounded by $s_{rj} \leq c_{j,\text{max}}$, where $c_{j,\text{max}}$ denotes the maximum logarithmic cardinality of the outer operand of join $j$.

**Proof.** By Eq. (9), an upper bound for $s_{rj}$ is given by

$$\log(\theta_r) + \log(\theta_j) \geq \log(\theta_r) + \log(\theta_j) + \log(\theta_j) s_{rj} = s_{rj}. \tag{12}$$

Since our model assumes uncorrelated predicates, the fraction of surviving tuples after joining multiple relations is given by the product of selectivities of all applicable predicates. Since $\text{Sel}(p) > 0$, an intermediate result contains at least one tuple, hence $c_j \geq 0$. The upper bound depends on the constant $\omega$, which needs to be sufficiently large to satisfy the constraint by activating $cto_{rj}$, but can otherwise be freely chosen.

Since we seek the smallest upper bound for $s_{rj}$, we next specify a lower bound for $\omega$. Following Eq. (8), this lower bound is given by $\omega \geq \omega_{\text{max}} - \log(\theta_r) \geq c_j - \log(\theta_r)$. Setting $\omega_{rj}$ to its lower bound, and inserting it into Eq. (12), produces $s_{rj} \leq c_{j,\text{max}}$. □

**Lemma 5.2.** The maximum logarithmic cardinality $c_{j,\text{max}}$ for the outer operand of join $j$ is given by $c_{j,\text{max}} = \sum_{i=1}^{j+1} \log(\text{Card}(i))$, where $\forall k < l : \text{Card}(k) \geq \text{Card}(l)$.

**Proof.** The logarithmic cardinality $c_j$ for join $j$ is given by $c_j = \sum_i \log(\text{Card}(i)) \cdot \text{tilo}_j + \sum_p \log(\text{Sel}(p)) \cdot \text{pao}_{pj}$. The cardinality may be reduced by applying predicates, since $0 < \text{Sel}(p) \leq 1$, making the logarithmic values negative. Since we consider the maximum cardinality, we set all variables $\text{pao}_{pj} = 0$, disregarding any predicates. The outer operand of join $j$ contains exactly $j + 1$ relations. The logarithmic intermediate cardinality is then maximised if the outer operand for join $j$ contains the first $j + 1$ relations out of a list of relations sorted in descending order by cardinalities. □

**Theorem 5.3.** Given $T$ relations, $P$ predicates and $R$ threshold values, an upper bound for the number of variables is given by $n \leq 2T \cdot (3P + R)(J - 1) + T + R \sum_{j=1}^{J-1} \left( \left\lceil \log_2 \left( \frac{c_{j,\text{max}}}{\omega} \right) \right\rceil + 1 \right)$, where $\omega$ is the discretisation precision for the continuous slack variables.

**Proof.** The number of binary variables is $n_{\text{pec}} + n_{\text{d}}$, where $n_{\text{pec}}$ is the number of problem-encoding variables given in Sec. 3.2, and $n_{\text{d}}$ counts slack variables for equality conversion.

First, we specify an upper bound for $n_{\text{pec}}$. As explained in Sec. 3.2, variables $\text{tilo}_j$ and $\text{pao}_{pj}$ are added $T \cdot J$ times, whereas variables $\text{pao}_{pj}$ and $\text{tilo}_j$ are added $J - 1$ times for $P$ predicates and $R$ threshold values. Depending on the concrete JO problem, we may be allowed to prune variables. A variable $\text{tilo}_j$ is unnecessary if the logarithmic cardinality of the outer operand for join $j$ can never exceed the logarithmic threshold value $\log(\theta_r)$. We therefore prune every variable $\text{tilo}_j$ if $c_{j,\text{max}} \leq \log(\theta_r)$. The number of variables when no pruning is possible then gives the upper bound $n_{\text{pec}} \leq 2T \cdot (3P + R)(J - 1)$. To specify an upper bound for $n_{\text{d}}$, consider that one binary slack variable is needed for each inequality constraint expressed by Eqs. (5),(6). As such, $T + 2P(J - 1)$ variables are required in these cases. In turn, multiple binary slack variables are needed to approximate continuous slack variables for constraints expressed by Eq. (8). Following from Lemma 5.1 and Eq. (10), an upper bound for the number of binary slack variables $n_{\text{d}}$ required to discretise all continuous slack variables for $J - 1$ joins and $R$ threshold variables is given by $n_{\text{d}} \leq R \sum_{j=1}^{J-1} \left( \left\lceil \log_2 \left( \frac{c_{j,\text{max}}}{\omega} \right) \right\rceil + 1 \right)$. Note that we specify an upper bound, since a constraint is only required if the corresponding variable $\text{cto}_{rj}$ has not been pruned. Considering the other inequality constraints, the upper bound for $n_{\text{d}}$ is given by $n_{\text{d}} \leq T + 2P(J - 1) + n_{\text{d}}$, which leads to the upper bound for the overall number of binary variables $n = n_{\text{pec}} + n_{\text{d}}$, or qubits, as

$$n \leq 2T \cdot (3P + R)(J - 1) + T + R \sum_{j=1}^{J-1} \left( \left\lceil \log_2 \left( \frac{c_{j,\text{max}}}{\omega} \right) \right\rceil + 1 \right).$$

□
6 DB-QPU CO-DESIGN FOR JOIN ORDERING

We now commence to deriving recommendations for future QPU designs customised towards serving as co-processors in databases, using insights gained from our experimental and formal analysis.

6.1 Qubit Recommendations

Given the formal analysis in Sec. 5, we first address problem scalability by deriving recommendations for qubit capacity. Fig. 4 thereby visualises upper qubit bounds, based on our formal analysis, for a variety of JO problems with up to 64 relations and different discretisation precisions. We choose cycle queries, as they require one additional join predicate compared to chain and star graphs.

The upper bound for the required logical qubits scales quadratically with the number of relations, the dominating scaling factor. Increase in discretisation precision has comparatively little impact on the upper bound compared to the number of relations, even if the difference in terms of qubits can reach more than 50% in some scenarios (top right). Nonetheless, as we have shown in the experimental analysis, this seemingly minor influence can have a decisive impact on the feasibility on current QPUs.

While qubit capacity for solving the largest problems considered with classical MILP solvers [75], where queries with 60 relations are very few years [34]. This provides an optimistic outlook w.r.t. the feasibility of solving JO problems on hypothetical improved QPU topologies, depending on the resulting circuit depths.

Increasing the qubit capacity (and thereby, solvable problem dimensions) based on a structural extension of the available connectivity graph, by adding additional qubit connections, and by using different quantum gate sets. To ascertain a representative selection of larger queries with an increasing amount of relations and varying query graph types (chain, star and cycle queries), we rely on the method of Steinbrunn et al. [72], using the query generator code by Trummer [73]. We consider problems with two threshold values, and minimal discretisation precision (i.e., $\omega = 1$).

Size Extrapolation. We consider baseline designs from IBM [36] (127 qubit Washington), IonQ [37], and Rigetti [66] (80-qubit Aspen-M), that is, their topologies and native gate sets. Similarly to IBM Q, Rigetti QPUs are based on superconducting qubits, whereas IonQ QPUs are based on trapped ions. This physical principle features stable, universally connected qubits, but slow gate times. Superconducting qubits offer less connectivity and stability, but faster gates [8, 58]. The repeating patterns of IBM and Rigetti topologies allow for straightforward extrapolation to larger numbers of qubits.

Density Extrapolation. Our experimental results suggest a large impact of connectivity limitations on JO feasibility, indicating great potential for improvement. As such, we augment existing topologies by adding new connections between previously non-adjacent qubits. We study the impact of adding new connections between previously non-adjacent qubits. A fully connected topology with $n$ qubits contains $N = n(n - 1)/2$ edges. When the baseline topology includes $M$ edges, we quantify the extended connectivity as $d = m/(N - M)$, where $m$ denotes the amount of added notes. $d$ lies in the interval $[0, 1]$, and interpolates between the baseline topology ($d = 0$) and a complete mesh ($d = 1$). We assume connections between non-adjacent qubits in topological proximity are more likely in future QPUs than between far-distant qubits. Instead of uniformly sampling from the set of all missing connections, we favour extending the connectivity between non-adjacent, but close qubits. Given the set of connections $C_\delta$ with distance $\delta$, we uniformly add connections sampled from $C_\delta$ starting with $\delta = 2$, until reaching the desired density, or all elements of $C_\delta$ have been added. Restart from $C_{\delta+1}$ in this case.

Results. Fig. 5 shows the depths of the quantum circuits for combinations of randomly generated join ordering problems, QPU architectures (varying topology and gate set), and transpilation methods. Concerning gate sets, we study the impact of transpilation onto the native set, which involves replacing any unsupported gate operation with an effectively equivalent chain of native operations, versus unrestricted gate sets, where we assume the QPU to natively support any possible gate operation. For density 0 (i.e., baseline topology), we notice a substantial increase in circuit depth for an increasing number of relations (notice the graph is in log scale!) that quickly exceeds NISQ capabilities. However, even very moderately increased densities (0.05 to 0.1) lead do much smaller circuit depths (up to one order of magnitude for the native gate sets). On IBM Q, relative differences in circuit depth are about identical between density 0 and 0.05, and 0.75 and 1—a albeit a fully meshed network with density 1 is obviously impossible using planar graphs.

Therefore, rather than improving qubit numbers beyond capacities sufficient for the bulk of practical queries, we heavily recommend improving qubit connectivity, since even a small amount of extra connections, which we deem achievable by future QPUs, can substantially impact the utility of QPUs for JO problems. Similar observations, albeit not as pronounced, hold for Rigetti.

6.2 Topology Recommendations

Based on our experimental results, which emphasized the need to address QPU limitations beyond just qubits, we now analyse the feasibility of solving JO problems on hypothetical improved QPU topologies, depending on the resulting circuit depths.

Scenarios. We extrapolate new topologies in three ways: By increasing the qubit capacity (and thereby, solvable problem dimensions) based on a structural extension of the available connectivity graph, by adding additional qubit connections, and by using different quantum gate sets. To ascertain a representative selection of larger queries with an increasing amount of relations and varying query graph types (chain, star and cycle queries), we rely on the method of Steinbrunn et al. [72], using the query generator code by Trummer [73]. We consider problems with two threshold values, and minimal discretisation precision (i.e., $\omega = 1$).
Figure 5: Circuit Depths for hypothetical future QPUs on varying join order instances. Experiments are performed with different query graph topologies, but these are not visualised since no relevant differences arise.

Transpiling a circuit onto the native gate sets (compared to unrestricted gates) also significantly increases depth as compared to a (hypothetical) unrestricted gate set for the Rigetti QPU, but does not significantly impact IBM Q. Nonetheless, judiciously expanding the set of supported native operations may also enhance problem feasibility of future QPUs as an alternative to improved topologies.

Circuit synthesis has gained recent attention for QPUs (see, e.g., Refs. [3, 12, 77]), for instance, for noise reduction [79] or approximate circuits [78]. Our experiments consider two (sufficiently mature) transpilation approaches, Qiskit [38] and tket [71]. While the scaling behaviour is essentially identical for increased connectivity density, we observe an overhead of typically 100% for tket over Qiskit for the superconducting platforms. Both produce comparable results for complete meshes (IonQ), so we conclude their capabilities are similar for boundary cases, but need to be carefully evaluated otherwise.

The IonQ platform features full connectivity between qubits as baseline, and is therefore not subjected to experiments with increasing density. The resulting circuit depths seem ideal compared to the superconducting platforms. Yet, the amount of qubits that can be supported by this physical technology is limited by the amount of individual ions that can be caught in a trap; current technology allows for tens of ions [24], and major improvements are not to be expected. Considering the predictions of Fig. 4, the advantages in circuit depth are therefore compensated by the lack of qubits.

7 RELATED WORK

Join ordering is among the most studied problems in database research [25, 31, 43, 45, 52–54, 56, 57, 72, 75, 81]. While join ordering on modern hardware, e.g., GPUs [48], is intensively being researched, no published work exists for JO with QCs, to the best of our knowledge. Very few endeavours address the use of QCs in databases, in stark contrast to the growing interest in quantum computing in other fields [14, 15, 39, 41, 60, 61, 63, 67, 76, 80].

DB transaction scheduling was studied by Groppe and Groppe [26], and Bittner and Groppe [10, 11]. The MQO problem was analysed by Trummer and Koch [74] for quantum annealing at the VLDB conference. They experimentally compare QA performance with classical approaches. Due to the hardware limitations, they had to focus on small-scale problems, where they achieved promising results by aligning specific problems to the available hardware. Fankhauser et al. [20] addresses MQO for gate-based QPUs.

Our method differs in several ways. JO and MQO problems are structurally different. While JO is concerned with deriving optimal join orders for a query plan, MQO seeks optimal plan combinations for multiple queries. We cannot reuse existing transformations for MQO, and propose a novel method to obtain a JO-QUBO formulation. The JO-QUBO transformation entails formulating JO as MILP, following Ref. [75], where Trummer and Koch solve JO with classical MILP solvers. However, we substantially adjust their formulation to arrive at a JO-QUBO encoding, and are the first to solve JO on QPUs, even gaining some quantum advantage.

Ref. [20, 74] analysed MQO on existing QPUs, without practical utility. For near-term utility, we derive DB-QPU co-design recommendations, and identify improvements tailored to JO.

8 DISCUSSION AND CONCLUSION

Quantum computing promises—grounded on theoretical insights and guarantees [8], but also based on first experimental results [5]—speedups for computational problems over classical approaches. The technology is apt for many aspects of database systems, including query optimisation. So far, the use of QCs for DB is extremely underexplored. Yet, our work is Janus-faced: On the one hand, we show that current NISQ-systems are far away from producing practical benefits, or handling realistically sized instances. This paints a more sober picture than initial optimistic evaluations of the technology. On the other hand, we show that with relatively minor adaptations, QPU-DB performance can be substantially enhanced. This prompts to use co-design approaches to create QPU-DB accelerators, given the commercial importance of databases.

Our results and predictions show that a multitude of factors influence the performance of QC approaches on DB problems, ranging from unusual problem formulations in QUBO form that massively diverge from traditional implementation techniques, to unfavourable scaling caused by subtle issues like discretisation precision, to
a complex interplay of physical implementation properties. This makes it impossible to delay QC integration into databases until sufficiently evolved hardware is available, but prompts the co-design of database accelerators. Simulating perfect quantum computers entails solving NP-hard problems, and adding the effects of noise and imperfections causes additional complexities. Co-design efforts should be based on step-wise empirical refinement leveraging expert knowledge from QC, DB, and systems engineering.

We lay the ground: Our approach enables the experimental exploration on two major classes of QCs, and identifies key factors that inhibit scalability and practical utility. We provide directions on designing QPUs favourable for JO. Nonetheless, many open research problems remain, from efficient circuit generation respecting noise, to more targeted extensions of topologies that transcend our semi-stochastic approach, to considering alternative information encoding schemes that might alleviate discretisation problems.

Yet such approaches are challenged by major physical and algorithmic obstacles, and require truly interdisciplinary research.

Contributions. MS (≈ ¾) and WM (≈ ¼) wrote the paper. SS suggested JO as problem of interest. WM and MS designed, and MS implemented and performed the experiments. WM and MS performed data analysis and visualisation.