Large-Scale Multiple Query Optimisation with Incremental Quantum(-Inspired) Annealing

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

Immanuel Trummer
Cornell University
Ithaca, NY, USA
itrummer@cornell.edu

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

Abstract

Multiple-query optimization (MQO) seeks to reduce redundant work across query batches. While MQO offers opportunities for dramatic performance improvements, the problem is NP-hard, limiting the sizes of problems that can be solved on generic hardware. We propose to leverage specialized hardware solvers for optimization, such as Fujitsu's Digital Annealer (DA), to scale up MQO to problem sizes formerly out of reach.

We present a novel incremental processing approach that combines classical computation with DA acceleration. By efficiently partitioning MQO problems into sets of partial problems, and by applying a dynamic search steering strategy that reapplies initially discarded information to incrementally process individual problems, our method overcomes capacity limitations, and scales to extremely large MQO instances (up to 1,000 queries). A thorough and comprehensive empirical evaluation finds our method substantially outperforms existing approaches. Our generalisable framework lays the ground for other database use-cases on quantum-inspired hardware, and bridges towards future quantum accelerators.

CCS Concepts

• **Theory of computation → Design and analysis of algorithms**; • **Hardware → Quantum technologies**.

Keywords

query optimisation, multiple query optimisation, quantum(-inspired) hardware

ACM Reference Format:
Manuel Schönberger, Immanuel Trummer, and Wolfgang Mauerer. 2018. Large-Scale Multiple Query Optimisation with Incremental Quantum(-Inspired) Annealing. In *Proceedings of SIGMOD (Conference acronym 'XX).* ACM, New York, NY, USA, 15 pages. https://doi.org/XXXXXXX.XXXXXXX

Conference acronym 'XX, Woodstock, NY

ACM ISBN 978-1-4503-XXXX-X/2018/06
https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

Processing ever-increasing computational loads and queries of growing size requires efficient data management techniques capable of maintaining high energy efficiency and computational throughput. One approach that is particularly attractive in highly parallel cloud DBMS deployments is to reduce query execution times by (1) considering queries as part of a *query batch* to be executed *in conjunction*, rather than in isolation, (2) identifying commonalities between queries within a batch, and (3) deriving a *globally optimal* execution plan for the entire query batch to avoid redundant result generation, thus reducing joint execution time. By balancing plan costs and cost savings opportunities, query optimisers can obtain speedups over the individually fastest plans, which may only feature few cost savings with other plans. This procedural outline is known as the classical problem of *multiple query optimisation* (MQO).

Despite its long history in data management research [42], comparatively few approaches have been proposed for MQO compared to other long-standing problems in the field. Yet, MQO remains uniquely well suited to address increasing computational demands particularly in cloud-centric query optimisation.

Most MQO methods target conventional *general-purpose* systems. Their performance stagnates with the end of Moore's law. Achieving substantial computational performance increases, particularly while maintaining energy efficiency, is an increasingly challenging effort. To address these growing concerns, recent trends in the database (DB) domain involve the analysis and use of *special-purpose* systems that are tailored to excel at specific computational tasks.

Quantum Hardware. Quantum processing units (QPU) count among the most highly anticipated special-purpose devices. Based on quantum bits (*qubits*), they are capable of exploiting quantum phenomena to obtain possibly exponential speed-ups over conventional systems. Trummer and Koch assessed their utility for MQO by deriving a suitable quantum annealer encoding [47].

Yet, contemporary quantum devices remain early-stage prototypes limited by various imperfections [20] that prevent their use on problems of industrially relevant sizes [38]. While Trummer and Koch identified the technological potential for *small* problem classes, the culminated impact of these limitations severely hampers the scalability aptness to larger problem sizes. *Qubit capacity*, as dominant limitation, substantially restricts scalability. Fig. 1 illustrates this issue for the original method by Trummer and Koch, which exceeds the capacity of annealers at the time of writing for problems featuring over 21 queries and moderate amounts of plans:

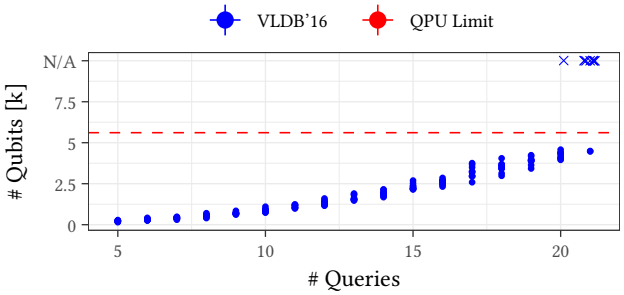

Figure 1: Qubit capacity requirements for the original quantum MQO method of VLDB'16 [47]. We show the required amount of qubits for a given number of queries for batches of ten problems with ten alternative execution plans per query. Crosses denote exceeded QPU capacity limits.

Their approach clearly does not scale to large MQO instances. Given hardware noise and other imperfections, solution accuracy quickly degrades even within the capacity limits of contemporary quantum systems, which limits quantum utility.

Quantum-Inspired Hardware. This sobering state prompts the study of *quantum-inspired systems*: Special-purpose accelerators whose design is informed by the workings of real quantum systems. Unaffected by the quantum HW imperfections, remarkable performance of quantum-inspired HW such as the Fujitsu Digital Annealer (DA) [2] has been demonstrated for large-scale problems that prove challenging for conventional general-purpose devices [39]. Such devices are thus ideal substitutes before actual quantum systems reach sufficient levels of maturity. Nonetheless, scalability on quantum-inspired hardware is also limited by their capacity to encode optimisation variables (*e.g.*, only 8,192 on DA hardware [2]).

Achieving Scalability. Still, the capabilities of quantum(-inspired) devices remain desirable to achieve our goal of efficiently processing ever-increasing data loads and overcoming the limits of general-purpose systems. In this paper, we therefore address the scalability limits of both, MQO approaches performed on conventional general-purpose systems, as well as the original quantum annealing method for MQO by Trummer and Koch [47], by deriving a novel *incremental multi-phase method* for quantum(-inspired) large-scale MQO that overcomes any device-specific capacity constraints.

Below, we briefly outline the salient characteristics of our hybrid approach that combines conventional and quantum(-inspired) HW.

(a) Problem Partitioning. To achieve our goal of overcoming capacity constraints, we apply a novel *problem partitioning* strategy tailored to MQO in the first phase of our method. Our approach transforms any MQO problem into a set of partial problems that can be individually processed within the capacity limitations of any target system. However, performing the partitioning phase on conventional HW would re-introduce the bottlenecks associated with general-purpose system. To avoid such slowdowns, our method involves *multiple uses* of quantum-inspired HW, respectively relying on it for (1) the actual MQO phase, yet moreover for (2) the partitioning phase, to derive ideal sets of partial MQO problems. Thereby, our approach achieves scalability aptness for large-scale problems beyond the capacity limits of any given quantum-inspired device, rendering MQO compatible with any existing and future system regardless of its individual size limitations.

(b) Dynamic Search Steering. While problem partitioning allows us to overcome capacity limitations, the resulting loss of global problem information may substantially degrade optimisation accuracy. The specific manner in which to process the resulting partial problems hence requires careful consideration. As such, we propose a novel *dynamic search steering* (DSS) strategy: By re-applying information initially discarded by the partitioning phase, we dynamically steer the search space exploration of still unsolved partial problems in accordance with the hitherto obtained partial solution. Rather than processing partial MQO problems independently, we thereby incrementally derive a total MQO solution tailored towards each partial problem. Doing so, our method not only maintains a high level of optimisation quality, but moreover substantially improves over competing partitioning approaches when solving very large problems, as we empirically show.

Contributions. In detail, our contributions are as follows:

(1) We derive a novel incremental processing strategy for quantum-inspired annealing that is tailored to MQO. Our method overcomes HW capacity limits by (1) partitioning MQO scenarios into sets of partial problems, and (2) incrementally processing them using our dynamic search steering (DSS) method, to account for initially discarded problem knowledge.
(2) We empirically assess the soundness of our novel incremental quantum-inspired annealing method for a variety of MQO scenarios. Firstly, our analysis includes a comprehensive parameter sweep that allows us to identify scenarios where the advantage of our method is most pronounced, and yields insights into special cases where the performance advantage deteriorates. To provide further indication on the general applicability of our method, our analysis moreover includes MQO scenarios derived from established query optimisation benchmarks, and demonstrates the advantage of our approach over both, established MQO baselines as well as alternative problem partitioning methods implemented by HW vendors. Most importantly, our method proves robust against the bulk of considered MQO scenarios, including extremely large problems featuring up to 1,000 queries, which prove intractable for competing methods.
(3) We benchmark the performance of two contemporary quantum and quantum-inspired HW types, including (1) the D-Wave Hybrid Quantum Annealer, and (2) the Fujitsu Digital Annealer. In our empirical analysis, we identify the most capable device for large-scale optimisation problems.
(4) Our complete algorithmic stack provides a framework to decouple quantum-inspired MQO from hardware constraints, rendering our method device-independent and compatible with all existing and future quantum-inspired annealing systems.

The remainder of this paper is structured as follows. Sec. 2 provides fundamentals on quantum(-inspired) annealing devices. Sec. 3 outlines MQO fundamentals and our considered problem model. In Sec. 4, we describe our novel incremental quantum(-inspired) annealing method for MQO in detail. Sec. 5 outlines our extensive empirical assessment of our method. Finally, we discuss related work in Sec. 6, and conclude in Sec. 7.

2 Quantum(-Inspired) Annealing

Quantum annealing constitutes a restricted variant of adiabatic quantum computation [16] and seeks to identify the ground state of an Ising Hamiltonian [15] that expresses a solution to an optimisation problem. It is not feasible to give a complete introduction to adiabatic quantum computation here. We thus refer the interested reader to Farhi *et al.* [16], and provide an overview of contemporary HW types implementing either pure quantum annealing, or a quantum-inspired variant. We discuss D-Wave's Hybrid Quantum Annealing in Sec. 2.2, and Fujitsu's Digital Annealing in Sec. 2.3.

2.1 Problem Encoding

Before discussing quantum(-inspired) annealing HW, we briefly outline the required problem encoding that is presupposed by all devices described in the following. Rather than running arbitrary code, quantum(-inspired) annealing requires problems to be encoded in accordance to the *quadratic unconstrained binary optimisation* (QUBO) formalism, which (1) only involves *quadratic interactions* between variables, (2) is *unconstrained*, therefore not supporting *explicit* constraints, (3) only features *binary* variables, and (4) encodes *optimisation* problems. Mathematically, they are given by the multivariate polynomial

$$f(\vec{x}) = \sum_i c_{ii} x_i + \sum_{i \neq j} c_{ij} x_i x_j,$$

where $x_i \in \{0, 1\}$ are variables, and $c_{ij} = c_{ji} \in \mathbb{R}$ coefficients, which may be interpreted as graph edges representing variable interactions between variable nodes. Physically, QUBO encodings correspond to *energy formulas*, and annealers seek their *minimum energy configuration*. Therefore, to encode optimisation problems for such devices, they have to be transformed in such a way that the minimum energy configurations of the resulting QUBO encoding correspond to ideal solutions for the original problem.

2.2 D-Wave Hybrid Quantum Annealing

We first consider the D-Wave Hybrid Quantum Annealer (HQA), which features both, use of quantum annealing, as well as steps performed on classical HW [23]. The system conducts an iterative optimisation that repeatedly queries quantum annealers on a limited representation of the overall problem input. These queries, representing sub-problems of limited size, are small enough to be solved on current quantum annealers and yield suggestions on search space regions to further explore. This hybrid framework allows the HQA to mitigate the size limitation of contemporary quantum annealers, which feature roughly 5,600 qubits to represent variables. By coordinating the overall optimisation via classical means, the HQA reduces the detrimental impact of noise, which perturbs computation on present-day quantum systems.

2.3 Fujitsu Digital Annealing

The Fujitsu Digital Annealer (DA) is a quantum-inspired device featuring special-purpose architecture tailored to solve QUBO-encoded problems based on an enhanced SA algorithm. The HW inherently supports problems featuring up to 8,192 optimisation variables. However, by performing problem partitioning steps [1], larger problems featuring up to 100,000 variables can be solved. While the DA's default partitioning method supports general QUBO-encoded problems, our novel incremental annealing method, described in detail in Sec. 4, is tailored to the characteristic properties of MQO. We analyse the advantage of our method in Sec. 5.

The DA enhances conventional SA in multiple ways. Firstly, it evaluates state updates for all *all decision variables in parallel* in each optimisation step, while conventional SA only considers flips for single variables at a time [2]. Doing so, the probability for state updates is substantially boosted [2]. In contrast to SA on classical HW, which yields a worst-case complexity of $O(N)$ for N variables, the DA's HW performs the required state updates in *constant time* ($O(1)$), regardless of the amount of associated variables. Further, the DA applies a *dynamic offset* escape method: If, upon assessing all variables, no state update is considered acceptable, the acceptance probability is increased in the next algorithmic step, thereby boosting the DA's aptitude to escape from local minima. For full details on the DA's algorithmic properties, we refer the reader to Aramon *et al.* [2].

Alternatives. In recent years, quantum-inspired alternatives to the DA have been made accessible. This includes the NEC Vector Annealer [33], which similarly performs a HW-augmented SA variant. While we have included the device in our initial assessment, we found both its optimisation accuracy and runtime performance to be dominated by the DA. Our empirical analysis of quantum-inspired HW in Sec. 5 hence focuses on the Fujitsu DA.

Accessibility. While our method is compatible with any future quantum device, we emphasise that our method does not require pure quantum computers. While inspired by quantum systems, systems such as the Fujitsu DA operate on purely classical means. Hence, like other conventional HW accelerators and high-performance computers, they are readily accessible via cloud services or direct HW purchases, and constitute very capable temporary substitutes for real quantum systems, as our experiments will show.

3 Formal MQO Model

Multiple query optimisation (MQO) seeks an ideal plan configuration minimising the accumulated execution cost for a batch of queries to be executed in conjunction [42]. Naively, a query optimiser may greedily select the individually fastest plan for each single query, failing to account for overlaps and cost savings opportunities with other query plans. Such plans may therefore be suboptimal when compared with plans selected by MQO, which are tuned to exploit common sub-expressions and *cost savings* opportunities between plans. Hence, by sharing intermediate query results, processing times for redundant result generation can be avoided, resulting in *lower processing times* compared to the naive selection of individually optimal query plans. The goal of MQO is hence to determine the selection of plans that yields the *globally minimal* execution costs, featuring the most efficient balance between individual plan costs and cost savings between plans.

[1] The specific algorithmic steps to partition QUBO-encoded problems are not fully disclosed by Fujitsu, and can thus not be further described here.

MQO involves two fundamental phases, and MQO methods may be classified in accordance to the steps they perform: The first category of MQO methods seeks to generate execution plans and to identify their common sub-expressions [9, 25]; in contrast, based on these results, the second category seeks to identify the globally ideal selection of execution plans, to minimise overall costs by exploiting common sub-expressions. The method proposed in this paper belongs in the second category, and hence presupposes the existence of common sub-expressions for plans whose ideal execution configuration is to be determined.

For the remainder of this section, we follow the formal NP-hard MQO problem model as presented by Trummer and Koch [47].

3.1 MQO Problem Input

Formally, the input for an MQO problem is given by (a) a set of queries Q to be executed, (b) a total set of execution plans P, and (c) a set of cost savings S. Each query with index $1 \leq q \leq |Q|$, where $|Q|$ denotes the size of Q, features an associated set of mutually exclusive execution plans $P_q \subseteq P$. Further, each cost saving $s_{p_1,p_2} \in S$ refers to a pair of different plans (p_1, p_2) not associated with the same query. Finally, each plan p_i, with $1 \leq i \leq |P|$, where $|P|$ denotes the size of P, features an associated execution cost $c_i > 0$. This cost may be reduced by $s_{p_i,p_j} \geq 0$, given cost sharing opportunities with another plan p_j.

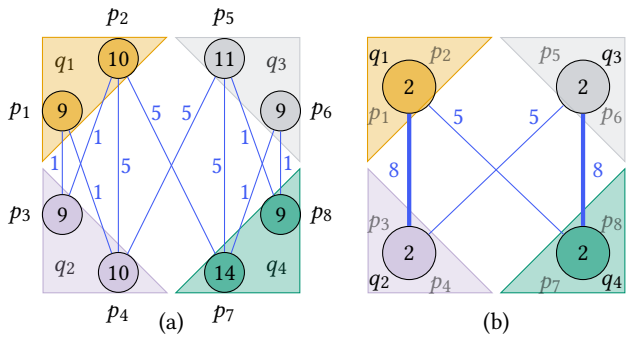

Figure 2: (a) MQO graph representing an MQO problem featuring four queries $(q_1, ..., q_4)$ and eight plans $(p_1, ..., p_8)$ in total. Nodes of varying colors represent plans associated with different queries. Edges represent cost savings between pairs of execution plans. Finally, nodes and edges are labeled with their respective costs and savings magnitudes.
(b) Partitioning graph compressing the MQO graph shown in (a), with nodes corresponding to queries labeled with the amount of associated plans, and edges labeled with the accumulated cost savings.

Each MQO problem may be represented by a corresponding *MQO graph* $G = (V, E)$, where each plan $p_i \in P$ is represented by a node $v_i \in V$, and each cost saving $s_{p_i,p_j} \in S$ is represented by a corresponding edge $e_{i,j} \in E$. Fig. 2 (a) depicts an MQO graph for a problem featuring four queries with two alternative execution plans per query. The MQO graph representation will prove useful for the partitioning phase of our approach.

We select the MQO input model outlined above due to its strong adherence to the required encoding formalism for quantum(-inspired) systems discussed below. Note, however, that the model is functionally equivalent to the alternative task-based model applied for prior MQO research [4, 11, 12], as discussed in [47].

3.2 MQO Solution

A solution for an MQO problem is given by a set of plans P_e, which includes those plans selected for execution. Thereby, $\forall q \in Q : |P_q \bigcap P_e| = 1$ must hold, *i.e.*, for each query, precisely one out of all alternative plans is selected. The size of the solution space is hence given by $N_{sol} = \prod_{q \in Q} |P_q|$ in general, or by $N_{sol} = PPQ^{|Q|}$ for a fixed amount of plans per query (PPQ). Finally, the total cost for a MQO solution is given by $C(P_e) = \sum_{p_i \in P_e} c_i - \sum_{\{p_i,p_j\} \subseteq P_e} s_{p_i,p_j}$, *i.e.*, the accumulated execution cost for all selected plans reduced by the amount of cost savings between plan pairs. An optimal MQO solution minimises this cost formula.

Example 3.1. To illustrate the MQO problem, let us consider the MQO scenario depicted in Fig. 2, featuring four queries with plans $P_{q_1} = (p_1, p_2), P_{q_2} = (p_3, p_4), P_{q_3} = (p_5, p_6)$ and $P_{q_4} = (p_7, p_8)$. We assume corresponding execution costs as $c_1 = 9, c_2 = 10, c_3 = 9, c_4 = 10, c_5 = 11, c_6 = 9, c_7 = 14, c_8 = 9$, and further assume cost savings as $s_{p_1,p_3} = 1$, $s_{p_1,p_4} = 1$, $s_{p_2,p_3} = 1$, $s_{p_2,p_4} = 5$, $s_{p_2,p_7} = 5$, $s_{p_4,p_5} = 5$, $s_{p_5,p_7} = 5$, $s_{p_5,p_8} = 1$, $s_{p_6,p_7} = 1$, $s_{p_6,p_8} = 1$.

Without considering result sharing opportunities, a query optimiser may greedily select the locally cheapest plan for each query, yielding the set of execution plans $P_{gr} = (p_1, p_3, p_6, p_8)$. While we obtain total costs $C(P_{gr}) = 9 + 9 + 9 + 9 = 36$ if the queries are executed in isolation, they can be reduced to 34 by considering cost savings $s_{p_1,p_3} = 1$ and $s_{p_6,p_8} = 1$. However, by considering such savings directly during during the plan generation, MQO can obtain the globally optimal set of execution plans $P_{opt} = (p_2, p_4, p_5, p_7)$, yielding total execution cost $C(P_{opt}) = 10+10+11+14-5-5-5-5 = 25$.

The exponential growth of the solution space, alongside its NP-hard nature, renders MQO a challenging problem when scaling to many queries. The industrial requirement to optimally process ever-increasing computational loads therefore prompts the use of novel HW such as quantum(-inspired) systems, which promise scalability and robustness through high architectural efficiency.

4 Incremental Quantum(-Inspired) Annealing

In this section, we present our incremental method for solving large-scale MQO problems with quantum(-inspired) annealing, using our novel partitioning and dynamic search steering strategies. We begin by discussing our problem partitioning phase in Sec. 4.1, and further outline our dynamic search steering phase in Sec. 4.2.

4.1 Problem Partitioning

The first phase of our method involves the partitioning of any given MQO problem into a set of partial problems that adhere to the capacity constraints of the quantum(-inspired) device. To avoid bottlenecks on conventional devices, we seek to not only rely on quantum(-inspired) annealers for the actual MQO phase, but moreover for the partitioning phase. However, processing the raw MQO problem on the device is clearly infeasible, since its insufficient

capacity prompts the partitioning phase in the first place. To still conduct this step on the device, we must hence transform the raw MQO problem into a *compressed form* adhering to capacity limits. To this end, we compress the input MQO graph into a *partitioning graph* of reduced size, such that conducting *graph partitioning* on the device yields the desired partial MQO problems.

4.1.1 Partitioning Graph Generation. To ideally minimise optimisation inaccuracies due to a loss of information by partitioning, we require an ideal set of partial MQO problems featuring the smallest possible amount of dependencies; that is, plans featured by different partial problems share the smallest possible amount of cost savings. To this end, the first step of our method involves the generation of a *partitioning graph*, which constitutes a compressed representation of the MQO graph, and features aggregated information to guide the subsequent graph partitioning step.

Formally, the partitioning graph $G = (V, E)$ contains a weighted node $v \in V$ for each query $q \in Q$ featured by the raw MQO problem, where the weight ω_{v_i} for a node v_i is given by the amount of alternative plans $|P_{q_i}|$ associated with query q_i. Further, G contains a weighted edge $e \in E$ for each pair of queries (q_i, q_j) sharing at least one cost saving between any of their respective plans, where $\omega_{e_{i,j}} = \sum_{s_{p_l,p_m} \in S, p_l \in P_{q_i}, p_m \in P_{q_j}} s_{p_l,p_m}$. Fig. 2 (b) illustrates the partitioning graph representing the MQO graph depicted in Fig. 2 (a).

Example 4.1. (cont'd) To illustrate the partitioning graph generation, we continue our MQO example for the scenario depicted in Fig. 2. Firstly, we derive the four nodes $V = (v_1, v_2, v_3, v_4)$, corresponding to the four queries featured by the problem, with equal weights $\omega_{v_1} = \omega_{v_2} = \omega_{v_3} = \omega_{v_4} = 2$ to represent the two plans featured by each query. We further add edges for each query pair sharing at least one cost saving: For (q_1, q_2), we add an edge $e_{1,2}$, and derive its weight $\omega_{e_{1,2}} = s_{p_1,p_3} + s_{p_1,p_4} + s_{p_2,p_3} + s_{p_2,p_4} = 1+1+1+5 = 8$ by summing over savings between plans featured by both queries. Likewise, we add edges $e_{1,4}$, $e_{2,3}$, and $e_{3,4}$ with weights $\omega_{e_{1,4}} = 5, \omega_{e_{2,3}} = 5$ and $\omega_{e_{3,4}} = 8$. In contrast, we do not require edges $e_{1,3}$ and $e_{2,4}$, as no savings are possible between plans featured by $(q1, q3)$ and $(q2, q4)$. We thus obtain the partitioning graph depicted in Fig. 2 (b).

4.1.2 Quantum(-Inspired) Graph Partitioning. Having outlined the generation of a partitioning graph, let us next consider the application of a weighted graph partitioning algorithm, which splits the set of graph nodes into two equally weighted subsets while minimising the accumulated weight of edges connecting nodes of the respective subsets. Since the node weights correspond to the amount of plans associated with the queries represented by each node, we obtain partial MQO problems featuring (1) two distinct, non-overlapping query subsets, and (2) approximately equally sized subsets of execution plans. We may recursively repeat this process by further partitioning our partial problems until none of them exceed the capacity limit of our target device. Further, by minimising the accumulated weight of edges connecting nodes of the respective subsets, a partitioning algorithm applied on our partitioning graph moreover minimises the accumulated magnitude of discarded savings represented by such edges. Thereby, our method yields valid partial MQO problems fulfilling our goal of maintaining high optimisation accuracy by minimising the loss of information.

Let us now consider a concrete method for processing the partitioning graph on a quantum(-inspired) device. As a precondition, the target device must feature variable capacity equal or greater than the amount of nodes in the partitioning graph, representing queries of the raw MQO problem. Then, we may derive a suitable mathematical encoding adhering to the QUBO formalism outlined in Sec. 2.1, which allows the use of quantum(-inspired) HW. For our method, we extend the base encoding for unweighted graph partitioning proposed in Lucas [27], to obtain an encoding for our weighted variant. Given our partitioning graph $G = (V, E)$ with node weight ω_{v_i} for node v_i and edge weight $\omega_{e_{i,j}}$ for edge $e_{i,j}$ connecting nodes v_i and v_j, and a spin variable $s_i \in \{-1, 1\}$ [2] for each $v_i \in V$, the encoding formula has the following partial terms:

$$H_A = \left(\sum_{i=1}^{N} \omega_{v_i} s_i \right)^2, \qquad H_B = \sum_{(u,v) \in E} \omega_{e_{u,v}} \frac{1 - s_u s_v}{2}.$$

Minimising H_A ensures *equal-sized, distinct query sets*, while minimisation of H_B minimises the loss of information resulting from partitioning, as respectively shown by Theorem 4.2 and Theorem 4.3 in the following.

Theorem 4.2. *Minimisation of H_A obtains two equal-sized, distinct sets of queries.*

Proof. Recall that the device seeks any variable configuration minimising H_A, which requires the sum inside the quadratic operation to ideally equal 0. Thus, to minimise H_A, the solver splits the set of spin variables into two subsets respectively featuring positive and negative spins, such that the accumulated weights (representing plan amounts featured by queries) are of equal magnitude and thus reciprocally eliminated. Otherwise, an energy penalty proportional to the size difference of the sets of plans is added. Since spins represent queries, minimising H_A obtains two distinct query sets featuring equal-sized amounts of plans. □

Theorem 4.3. *Minimisation of H_B minimises the magnitude of cost savings discarded by the partitioning.*

Proof. Minimising H_B requires $s_u = s_v$ for as many pairs of nodes (v_u, v_v) connected by edges in E as possible. Otherwise, if $s_u \neq s_v$, edge weight $\omega_{e_{u,v}}$ is added for any edge connecting nodes of unequal partitions. Thus, minimising H_B yields node subsets where the accumulated edge weight between nodes of either set, and hence the savings magnitude discarded by partitioning, is minimised. □

Example 4.4. (cont'd) We continue our MQO example for the scenario depicted in Fig. 2, by demonstrating how quantum solvers minimise the encoding discussed above. We begin by considering H_A, whose minimisation yields balanced MQO partitions $part_1$ and $part_2$. For instance, the annealer may obtain $part_1 = (q_1, q_2)$ and $part_2 = (q_3, q_4)$, where the corresponding spin variables $s_1 = 1$, $s_2 = 1$, $s_3 = -1$ and $s_4 = -1$, with weights corresponding to the two plans for each query, yield energy $H_A = (2 \cdot 1 + 2 \cdot 1 + 2 \cdot -1 + 2 \cdot -1)^2 = 0$. The same energy is obtained for any other configuration representing a split into equal-sized partitions. In contrast, an imbalanced split $part_1 = (q_1, q_2, q_3)$ and $part_2 = (q_4)$ yields energy

[2] This encoding is functionally equivalent to the model featuring a binary variable domain outlined in Sec. 2.1 [5].

$H_A = (2 \cdot 1 + 2 \cdot 1 + 2 \cdot 1 + 2 \cdot -1)^2 = 16$, while failure to apply any partitioning with $part_1 = (q_1, q_2, q_3, q_4)$ and $part_2 = \emptyset$ obtains $H_A = (2 \cdot 1 + 2 \cdot 1 + 2 \cdot 1 + 2 \cdot 1)^2 = 64$.

Thus far, energy minimisation obtains any balanced set of partitions. However, we seek those configurations that minimise the loss of information, *i.e.*, the accumulated savings between queries belonging to different partitions, expressed by the weight $\omega_{e_{i,j}}$ for any pair of queries (i, j). Hence, we further consider the energy induced by H_B: For the configuration $part_1 = (q_1, q_2)$ and $part_2 = (q_3, q_4)$, we obtain the minimum energy $H_B = \omega_{e_{1,2}} \cdot 0 + \omega_{e_{1,4}} \cdot 1 + \omega_{e_{2,3}} \cdot 1 + \omega_{e_{3,4}} \cdot 0 = 8 \cdot 0 + 5 \cdot 1 + 5 \cdot 1 + 8 \cdot 0 = 10$. In contrast, $part_1 = (q_1, q_4)$ and $part_2 = (q_2, q_3)$ obtains $H_B = 8 \cdot 1 + 5 \cdot 0 + 5 \cdot 0 + 8 \cdot 1 = 16$, while $part_1 = (q_1, q_3)$ and $part_2 = (q_2, q_4)$ yields $H_B = 8 \cdot 1 + 5 \cdot 1 + 5 \cdot 1 + 8 \cdot 1 = 26$.

In our example, an annealer may attempt to further reduce the H_B energy from 10 to ideally 0, by breaking the balance established by H_A. However, as shown above, H_A penalises imbalances with an even larger energy penalty of at least 16, rendering such attempts fruitless in our example. However, in general, imbalanced configurations may yield lower energies depending on the specific properties of the MQO problem.

Such imbalances are tolerable if they do not hamper subsequent processing steps, and may be desirable to prevent an unfavourable distribution of queries that yields a large loss of information. They become problematic if partial problems violate capacity constraints of the target device. Thus, to avoid excessive imbalances and potentially ineffective partitions caused by strict balancing, we apply a two-folded approach: Firstly, a Lagrange multiplier ω_A augments H_A, such that the positive energy penalty induced by validity violations in H_A always outweighs any negative energy benefit obtainable for H_B as a result of violations. This guarantees balanced partitions in minima of $H = \omega_A H_A + H_B$ (Theorem 4.5 below). Secondly, a post-processing mechanism described in Sec. 4.1.3 further reduces loss of information by effectively re-arranging query partitions, while giving us full control over minimum partition sizes to achieve a sufficient size reduction.

Theorem 4.5. *Minimising $H = \omega_A H_A + H_B$ obtains balanced query partitions if $\omega_A = \max_{q_i \in Q}(\sum_{q_j \in Q, q_j \neq q_i} \omega_{i,j})$.*

Proof. We begin by considering the minimisation of H_A for a set of queries Q, which yields equal-sized query partitions as per Theorem 4.2. Consider the combined encoding $H = H_A + H_B$ on a query q_i. While H_B yields energy 0 for each query $q_s \in Part(q_i)$, *i.e.*, if q_i is featured by the same partition $Part(q_i)$ as q_i, it induces a further penalty of $\omega_{e_{i,j}}$ for each query $q_j \notin Part(q_i)$ (recall that $\omega_{e_s i,j}$ denotes an edge weight in the partitioning graph). As annealers minimise the global energy $H = H_A + H_B$, partition balance may be violated by shifting partitions for a query q_i, if the magnitude of the positive energy penalty p thus induced by H_A is smaller than the negative energy benefit $b = \sum_{q_j \notin Part(q_i)} \omega_{e_{i,j}}$ induced by H_B.

An upper bound $b_{max} = \max_{q_i \in Q}(\sum_{q_j \in Q, q_j \neq q_i} \omega_{i,j})$ over all queries denotes the maximum accumulated cost savings shared between one query and all others. Augmenting H_A as $H = \omega_A H_A + H_B$ with $\omega_A = b_{max} = \max_{q_i \in Q}(\sum_{q_j \in Q, q_j \neq q_i} \omega_{i,j})$ ensures that the energy penalty for violating H_A outweighs any benefit of sacrificing the partition balance induced by H_B. □

4.1.3 Post-Processing. While minimising the QUBO encoding discussed above ensures balanced query partitions, controlled imbalances can be desirable for queries that cannot be evenly divided without a substantial loss of information. We hence propose the post-processing algorithm featured in Algorithm 1, to obtain more fine-grained query distributions.

Algorithm 1 Post-Processing Partitioning Results

```
1:  function PostProcess(PG, part1, part2, numParses, minSize)
2:      for i ← 1 to numParses do
3:          for query ∈ part1 do
4:              if |part1| == minPartSize then
5:                  break
6:              // Calc. the accumulated savings to either partition
7:              // in accordance with partitioning graph PG
8:              p1Conformance ← AccSavToP1(PG, query, part1)
9:              p2Conformance ← AccSavToP2(PG, query, part2)
10:             if p1Conformance < p2Conformance then
11:                 part1.remove(q)
12:                 part2.append(q)
13:     return (part1, part2)
```

The goal of our post-processing method is to shift queries from the first given partition *part1* to *part2* if it can thus achieve a reduction in the magnitude of discarded cost savings. For each query contained in *part1*, our method determines both, the conformance of the query to its own assigned partition *part1*, as well as the second partition *part2*. The conformance values correspond to its respective accumulated cost savings to queries contained in *part1* (*AccSavToP1*) or *part2* (*AccSavToP2*), based on the edge weights of the partitioning graph *PG*. If a query contained in *part1* thus features a higher conformance to queries of *part2*, and may only have been assigned to *part1* by the annealer to obtain equal-sized partitions, it is accordingly reassigned to *part2*, to further reduce the loss of information. Our method terminates once *part1* has been reduced to the specified minimum size, and can hence be parameterised in accordance to the capacity constraints of any target device.

Further, as the conformance values dynamically change with every query shifted to *part2*, our post-processing method repeats this process for every query in accordance with the specified number of parses (*numParses*). Hence, while a query may not be reassigned during the first parse, the partition swap of strongly associated queries may cause its shift during subsequent parses. Naturally, the outcome of our method depends on which of two partitions is specified as *part1*, thus having queries removed, and which is specified as *part2* and is accordingly assigned further queries. As such, our post-processing method is best performed on both possible assignments, to select the best among both results.

4.2 Incremental Optimisation with DSS

Having obtained partial MQO problems adhering to the size limitations of our target device, we next outline the actual optimisation phase of our method. We begin by contrasting two possible strategies to optimise the set of partial problems.

Parallel Optimisation. The first processing option involves optimising each partial MQO problem *independently*, which obtains partial solutions optimised for each individual partial problem. By merging all partial solutions, we then obtain a total solution to the original MQO problem. Independently processing each problem allows parallel execution on multiple annealers, which accordingly reduces the total optimisation time. However, by disregarding dependencies between partial problems, solution quality may be severely hampered, leading to suboptimal performance in contrast to processing strategies that account for such dependencies. Indeed, we propose one such method in the following, and assess its advantage over the naive, parallel processing variant in our experiments discussed in Sec. 5.

Example 4.6. (cont'd) We show the parallel optimisation strategy by continuing our MQO example for the scenario depicted in Fig. 2. Following Example 4.4, the partitioning phase yields two partitions $part_1 = (q_1, q_2)$ and $part_2 = (q_3, q_4)$. By applying MQO on each partition independently, we obtain the solutions $sol_1 = (p_2, p_4)$ with costs $C(sol_1) = c_2 + c_4 - s_{p_2,p_4} = 10 + 10 - 5 = 15$, and $sol_2 = (p_6, p_8)$ with costs $C(sol_2) = c_6 + c_8 - s_{p_6,p_8} = 9 + 9 - 1 = 17$. By merging sol_1 and sol_2, we obtain the total solution $sol_{ttl} = (p_2, p_4, p_6, p_8)$, and accounting for all original savings yields costs $C(sol_{ttl}) = c_2+c_4+c_6+c_8-s_{p_2,p_4}-s_{p_6,p_8} = 10+10+9+9-5-1 = 32$. By failing to account for the discarded savings s_{p_2,p_7} and s_{p_4,p_5}, the parallel processing strategy yields a more expensive solution when compared with the optimal solution with costs 25 derived in Example 3.1.

Incremental Optimisation. By independently processing each partial MQO problem, we obtain partial solutions that may be optimal for each of the partial problems. Yet, much like how choosing the locally optimal execution plan for single queries obtains globally suboptimal solutions which fail to exploit cost savings between plans (thus motivating MQO in the first place), optimising partial MQO problems independently yields suboptimal solutions for the original MQO problem, neglecting to account for and exploit dependencies between the partial problems. Independent processing of partial problems thus fails to capture cost savings between plan pairs featured by different partial problems, which have been discarded by the problem partitioning step.

While it is not possible to account for those cost savings *during* the optimisation of a partial MQO problem, we may consider them *inbetween* the optimisation steps of the partial problems. We thus propose an *incremental optimisation method* which gradually builds a total solution while coordinating the successive optimisation of partial problems. The outline of our incremental method is featured in Algorithm 2. Most importantly, we apply a *dynamic solution steering* (DSS) strategy, detailed in Algorithm 3, to *tune* the optimisation of still unsolved partial problems in accordance with the cost savings and information extracted from already obtained partial MQO solutions. Thus, our method incrementally builds a global MQO solution by gradually steering the partial problem processing towards a common objective, rather than independent partial goals.

Following Algorithm 2, we begin by encoding each partial problem based on the encoding method by Trummer and Koch [47], to obtain a QUBO encoding for quantum(-inspired) annealing. We next determine the best among all obtained solutions. Rather than selecting the solution with the lowest costs for the individual partial problem, we identify the best solution as the one that minimises costs w.r.t. the *current total solution* encompassing all plans selected thus far, and add it to our total solution *ttlSol*, such that *ttlSol* contains the complete solution after processing all partial problems.

Algorithm 2 Incremental Optimisation

```
1:  function IncrementalOptimisation(partProblems)
2:      // Initially, our total solution ttlSol is empty
3:      ttlSol ← ∅
4:      while partProblems ≠ ∅ do
5:          // Get next partial problem to process
6:          prob ← partProblems.pop()
7:          // Encode partial problem
8:          p_e ← EncodeProblem(prob)
9:          // Solve partial problem on quantum(-inspired) device
10:         partSols ← solver.optimise(p_e)
11:         // Determine best partial solution w.r.t. ttlSol
12:         bestPartSol ← BestIntSol(ttlSol, partSol)
13:         // Add partial solution to total solution
14:         ttlSol ← ttlSol ∪ bestPartSol
15:         // If at least one problem remains unsolved
16:         if partProblems ≠ ∅ then
17:             // Update remaining problems with DSS
18:             partProblems ← DSS(ttlSol, partProblems)
19:     return ttlSol
```

Algorithm 3 Dynamic Search Steering

```
1:  function DSS(intSol, partProblems)
2:      for prob ∈ partProblems do
3:          // Consider each discarded saving for problem
4:          for s ∈ prob.discSavings do
5:              // Consider each plan in the unsolved problem
6:              for q ∈ prob.queries do
7:                  for plan ∈ q.plans do
8:                      // For each plan in intermed. solution intSol
9:                      for selPlan ∈ intSol.plans do
10:                         // Verify if discarded saving applies
11:                         if (plan, selPlan) ⊆ s.plans then
12:                             // Reduce costs by disc. saving
13:                             plan.cost ← plan.cost − s.val
14:     return partProblems
```

Finally, before repeating the process for the next problem, we adjust each still unsolved partial problem in accordance to our dynamic search steering (DSS) strategy. Following Algorithm 3, our method *updates* the execution costs of all plans featured by unsolved problems, such that they reflect the *true costs* when accounting for already selected plans stored in the current intermediate solution *intSol*. Thus, plan costs of unsolved problems are reduced by each cost saving shared with already selected plans. Thereby, information initially discarded during the problem partitioning phase is reintegrated into the optimisation process by dynamic plan cost adjustments. We thus steer the optimisation of individual problems

towards a common direction, which substantially boosts optimisation quality in comparison to the parallel processing method described above, as we will show in Sec. 5.

Example 4.7. (cont'd) We continue our running example by demonstrating our incremental processing strategy with DSS for the partitions $part_1 = (q_1, q_2)$ and $part_2 = (q_3, q_4)$ obtained in Example 4.4. As before, we begin by applying MQO for $part_1$, which yields $sol_1 = (p_2, p_4)$ with costs $C(sol_1) = c_2 + c_4 - s_{p_2,p_4} = 10 + 10 - 5 = 15$. However, rather than optimising $part_2$ independently, we next apply our DSS method, to steer the optimisation of $part_2$ in accordance to the partial solution sol_1: Exploiting the knowledge that p_2 and p_4 have already been selected for $part_1$, we update the costs of plans in $part_2$ based on the initially discarded cost savings s_{p_2,p_7} and s_{p_4,p_5}. Since sol_1 already features p_2, s_{p_2,p_7} will apply if p_7 is selected for $part_2$. Hence, reducing $c_7 = 14$ by $s_{p_2,p_7} = 5$, we derive the updated costs $c_{7_{DSS}} = 9$. Likewise, we reduce $c_5 = 11$ by $s_{p_4,p_5} = 5$, knowing that p_4 has already been selected, and obtain $c_{5_{DSS}} = 6$. Finally, successfully applying MQO for $part_2$ now yields the solution $sol_2 = (p_5, p_7)$ with costs $C(sol_2) = c_{5_{DSS}} + c_{7_{DSS}} - s_{p_5,p_7} = 6+9-5 = 10$, and merging sol_1 and sol_2 obtains the optimal solution $sol_{opt} = (p2, p4, p5, p7)$ with costs 25, as determined in Example 3.1 for the unpartitioned case.

Post-Processing. While all our considered annealers consistently obtain valid MQO solutions in our analysis below, validity is not guaranteed. For devices yielding invalid solutions, post-processing may, for instance, choose the plan with the lowest costs w.r.t. valid query plans if multiple plans are selected for a query, and likewise select the best among all possible plans if no plan was selected.

5 Experimental Analysis

We now conduct a systematic empirical analysis to comprehensively evaluate the aptness of our incremental quantum(-inspired) annealing method for large-scale MQO.

5.1 Experimental Setup

We seek to identify the most suitable algorithm for large-scale MQO, by analysing both established MQO approaches, as well as quantum(-inspired) annealing methods, augmented with our novel incremental annealing strategy. For MQO problems of limited size, methods such as the A-* algorithm [10, 42, 43] may be used to determine optimal solutions. However, their required optimisation time increases exponentially when scaling up problem dimensions [4]. This renders such approaches unsuitable for our experiments, which seek to evaluate scalability aptness for large-scale problems.

We select a set of baselines known to scale to moderate and large problem dimensions (hill climbing (HC) [12] and the genetic algorithm [4]). Standard simulated annealing (SA) solves QUBO-encoded problems on conventional HW, and provides a baseline to assess the advantage of quantum(-inspired) special-purpose HW. For this, we consider the D-Wave Hybrid Quantum Annealer (HQA) [23] and the Fujitsu Digital Annealer (DA) [2]. Processing methods include (1) the *default* method offered by the annealer to partition large-scale problems, (2) our parallel annealing strategy that processes each partitioned problem independently, and (3) our novel incremental quantum(-inspired) annealing strategy, using DSS to exploit dependencies between all partitioned problems. The experimental analysis thus involves eight approaches: ***HC***: Hill climbing algorithm by Dokeroglu *et al.* [12]; ***Genetic***: Genetic algorithm proposed by Bayir *et al.* [4]; ***SA (Default)***: Simulated Annealing on classical HW; ***SA (Incremental)***: Simulated Annealing on classical HW, using our incremental annealing strategy with DSS; ***HQA***: Hybrid Quantum Annealing, using our incremental annealing strategy with DSS; ***DA (Default)***: Quantum-Inspired Digital Annealing, using default partitioning as supported by Fujitsu; ***DA (Parallel)***: Quantum-Inspired Digital Annealing, using our parallel annealing strategy; ***DA (Incremental)***: Quantum-Inspired Digital Annealing, using our incremental annealing strategy with DSS. For *HC* and *Genetic*, we use performance-enhanced[3] Java code by Trummer [46] wih OpenJDK 18.0.2. For *Genetic*, we use the Java Genetic Algorithms Package [29] 3.4.4, and population sizes 50 and 200 with default values for all parameters. We consider the best results obtained within 300s. For *SA (Default)*, we run an implementation in python (version 3.8.16) using dwave-neal 0.6.0 with 1,000 annealing iterations (default), and reduced amounts (to ascertain an identical overall amount of annealing iterations) for each MQO partition when performing our incremental annealing strategy. We perform 16 runs for each problem yielding one MQO solution. All baselines are executed on an octa-core AMD Ryzen 7 PRO 5850U CPU with 32 GB of DDR4 RAM.

Next, we consider parameter settings for our quantum(-inspired) annealing devices, where the user may specify either (1) the number of annealing iterations (algorithmic steps), or (2) the maximum optimisation time depending on the device. The D-Wave HQA specifies minimum optimisation times depending on problem dimensions, which we choose for each problem. For the DA, we select the minimum of 16 runs, with 20s of optimisation time per run. When performing our incremental annealing strategy, we reduce annealing iterations and optimisation times for each partition accordingly, such that the overall annealing time remains constant. In addition to processing the balanced partitions yielded by the QUBO minimisation, we moreover apply our partitioning post-processing method with four parses, and select the best results. Finally, given our limited budget and relatively large HQA minimum optimisation times, we are unable to conduct HQA experiments for MQO problems beyond 500 queries. To keep the monetary access fees manageable, we only apply HQA with our incremental annealing variant, and restrict our HQA experiments to a single problem instance per class, which is sufficient to determine the HQA's performance ranking compared to our remaining algorithms.

5.2 Parameter Sweep

The first part of our empirical evaluation consists of a comprehensive parameter sweep over input dimensions to identify the most suitable algorithms for a broad variety of MQO scenarios.

5.2.1 Problem Generation. To assess scalability of all approaches, we generate problems with varying amounts of queries $|Q|$ and plans per query (PPQ), corresponding to MQO graphs of increasing *node amounts*. This captures both determinants of MQO solution space size $N_{sol} = PPQ^{|Q|}$. For MQO graph *edges*, corresponding to

[3]The optimised baseline implementations can be found in our reproduction package [41].

cost savings between query plans, we vary (1) *cost savings density* [4] to see the effect of increasing amounts of cost savings, and (2) *distribution* of cost savings to assess the impact of varying graph shapes, which can substantially impact partitioning methods.

To generate varying graph shapes, we randomly distribute all queries into distinct *query communities*, where single communities semantically imply larger overlaps between queries, and hence a greater potential for cost savings opportunities. Accordingly, for sets of plans associated with queries included in the same community, we sample cost savings densities from increased intervals (with $[0.05, 1]$ as our largest interval, allowing for the maximum density of 100%), whereas the cost savings density between plan pairs separated into different communities is set to 0.05, and is hence ensured to be lower than community densities.

Finally, cost savings between plans are sampled from $[1, 10]$, while plan costs are uniformly sampled from $[1, 20]$, and further increased by an offset to account for increasing cost savings between plan pairs when scaling up problem sizes. Offsets are selected such that the absolute costs yielded by the best-performing algorithm remain roughly constant as problem dimensions increase. Thus, normalised solution costs reflect the evolution of gaps in absolute cost values yielded by our considered algorithms, and our problem generation ensures that varying optimisation performance yields pronounced differences in solution quality. While normalised costs depend on the chosen cost offsets, the relative performance ranking between algorithms is invariant w.r.t. offset selection. All generated problems can be found in our reproduction package [41].

Our parameter sweep includes 140 distinct MQO problems, with three generated instances for each class. Combined with 60 problems generated from query optimisation benchmarks, we consider 200 problems representing a large variety of possible scenarios.

5.2.2 Scalability Robustness. To understand the effect of increasing solution space size, Fig 3 depicts results for scenarios with increasing amounts of queries and *PPQ*.

Conventional Heuristics. Firstly, we consider conventional heuristics, including both, the genetic algorithm and hill climbing (HC). Even for our smallest MQO scenario (250 queries and 20 *PPQ*), genetic normalised solution costs slightly exceed 3, and hence feature a cost increase of over 200% over the best solution found by any algorithm. In contrast, normalised HC costs are still higher at 3.73. While already substantial for our smallest problems, the performance gap between the conventional heuristics and the best-performing algorithms becomes still more pronounced as the number of queries increases. For 20 *PPQ* and 750 queries, normalised costs range as high as 18 for the best genetic algorithm configuration, and beyond 20 for HC, whereas neither method yields normalised costs below 20 for 1,000 queries. Compared to the best performing algorithm that we discuss below, the conventional heuristics demonstrate a lack of robustness against large-scale MQO problems.

Simulated Annealing. We next consider results for SA, which performs algorithmic steps akin to DA, albeit on conventional HW. This provides a baseline to assess the advantage of quantum(-inspired) special-purpose systems. We find a pronounced advantage, as SA yields a performance comparable to conventional baselines, and even fails to match the performance of the genetic algorithm in most scenarios. While SA costs are already substantial for MQO scenarios featuring up to 500 queries, the method obtains solutions approximating normalised costs of 20 for 750 queries. Like the conventional baseline heuristics, SA hence fails to match the performance of competing methods for large-scale MQO.

Hybrid Quantum Annealing. We continue by discussing results for HQA as the first among our quantum(-inspired) MQO algorithms. Compared to most competing methods with the exception of DA, HQA produces lower solution costs, in particular for scenarios featuring 500 queries: For 40 *PPQ*, HQA yields normalised costs of 6.13, whereas the competitors discussed so far obtain costs that approximately range between 8 and 10. Yet, normalised solution costs remain relatively large when contrasted with DA performance. While the further increased minimum optimisation times required by D-Wave and limited budget prevent us from conducting experiments beyond 500 queries, we may conclude that HQA performance beats the conventional competitors, yet fails to obtain the same level of robustness against large MQO solution spaces as the DA.

Digital Annealing. Finally, we consider results obtained by the Fujitsu DA, where we contrast various processing methods, including the default processing as supported by the DA, as well as both, the parallel processing strategy and our incremental DSS-guided strategy. For all MQO scenarios depicted in Fig. 3, we observe that all processing strategies manage to yield significantly lower solution costs compared to any of our other considered MQO methods, which demonstrates the general aptness of the DA for large-scale optimisation. Further contrasting the three processing strategies, our incremental DA method can be identified as the best among all methods, obtaining normalised optimal solutions in all scenarios shown in Fig. 3. As a general tendency, performance gap between our incremental method and both, default as well as parallel processing, becomes more pronounced as the solution space increases, including both (1) growing query amounts and (2) increasing amounts of *PPQ*. For our largest problems featuring 1,000 queries and 40 *PPQ*, normalised costs for default and parallel processing reach up to 7.3 and 9.7 respectively, exceeding incremental solution costs by a substantial amount, which demonstrates the aptness of our incremental processing strategy.

While the trend towards growing performance gaps alongside an increasing solution space holds for most scenarios, we can observe some outliers where the difference between default and incremental processing is less pronounced (*e.g.*, for one problem featuring 1,000 queries and 40 *PPQ*). While these may be attributed to the stochastic nature of the DA's search algorithm, they may moreover result from data characteristics influenced by other, still unconsidered MQO input dimensions. As such, we next further contrast the performance of our varying processing strategies for other dimensions.

5.2.3 Increasing Numbers of Communities. Having identified the DA as the most capable MQO approach, we commence to study performance differences between the DA's default processing method and our DSS-guided incremental strategy. In Fig. 4, we consider

[4] The cost savings density is given by the fraction of cost savings featured by an MQO instance over all possible cost savings.

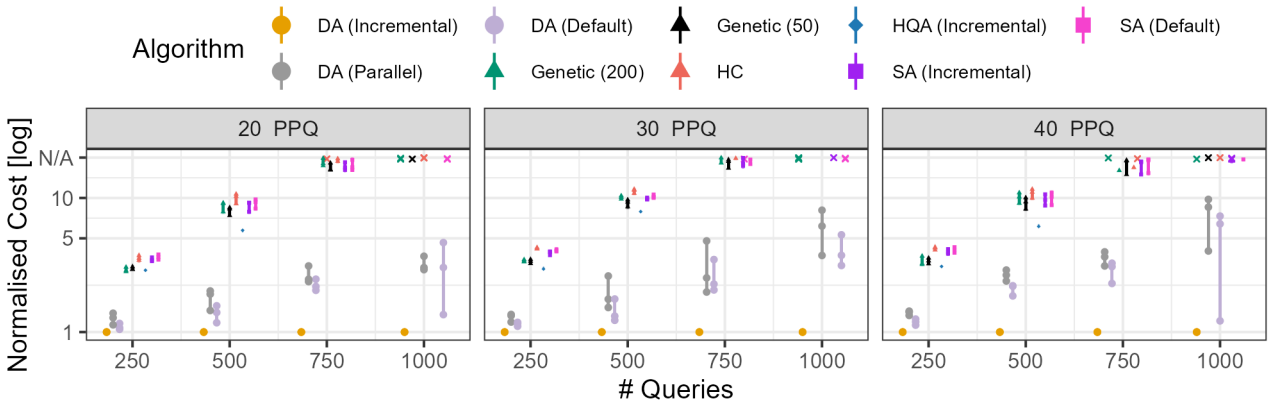

Figure 3: Normalised solution costs (relative to overall best solutions) for instances with varying amounts of queries and plans per query (PPQ), and four query communities of varying sizes. We sample community densities from [0.05, 1]. Range lines visualise minimal and maximal normalised costs for a set of instances; individual solutions are given by small dots. Crosses with value N/A represent prohibitively large costs ≥ 20 (slightly jittered horizontally and vertically to resolve outlier count).

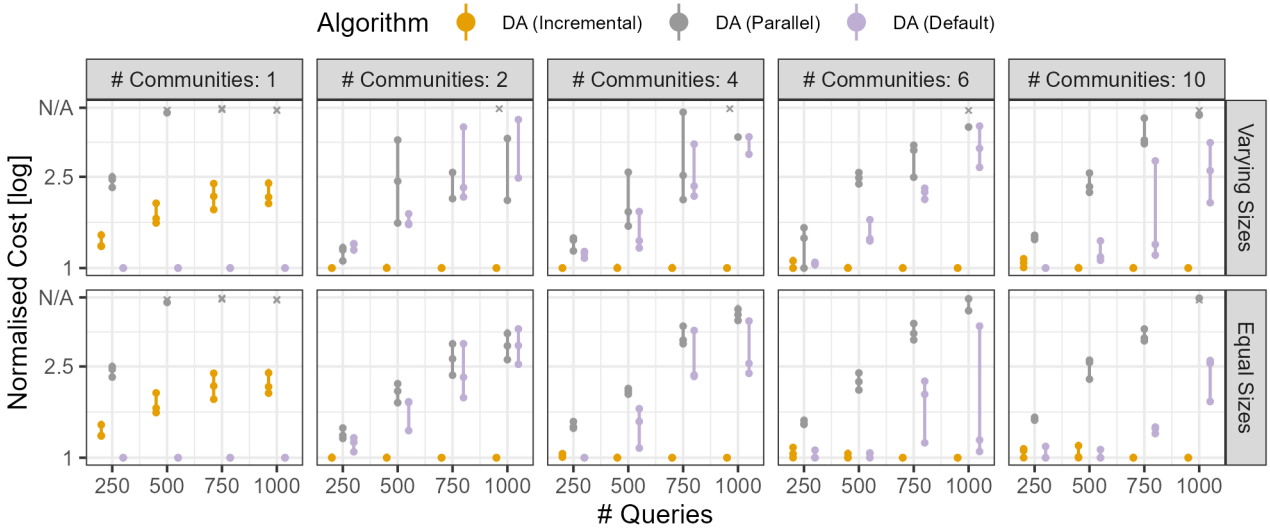

Figure 4: Normalised solution costs for problems with 30 PPQ for varying numbers of queries and query communities; community densities are sampled from [0.05, 1]. Instances exhibit varying (top row) and equal (bottom row) community sizes. Costs ≥ 5 are mapped to N/A; other details follow the presentation in Fig. 3.

increasing numbers of query communities, and compare scenarios with equally sized communities against variations in size.[5]

We begin with a single query community where plans of all query pairs are equally likely to feature cost savings. Our incremental processing method provides no advantage over the DA's default processing, as no structural characteristics can be exploited by our targeted partitioning and DSS methods. While incremental processing performance deteriorates for unstructured and uniform cases, they are unlikely for realistic queries, reinforced by results for conventional query optimisation benchmarks below.

Once MQO scenarios exhibit apt characteristics (multiple distinct communities of structurally similar queries likely to feature cost savings between plan pairs), our incremental method identifies and exploits these, and consistently obtains cheaper solutions than DA's default processing and other algorithms. Similarly to scenarios with varying *PPQ* amounts, parallel processing yields substantially more costly solutions. It is typically inferior to both, the DA's default processing and our incremental method, which demonstrates the benefit of our DSS approach.

By further contrasting scenarios with equal-sized communities and cases with varying community sizes, we observe a further trend: While the advantage of our method over the DA's default processing is most pronounced for communities of varying size, it is weaker for equal-sized communities, in particular for scenarios

[5] As we focus on processing strategies, we omit results for the remaining MQO approaches. Their solution costs relative to the DA remain substantial across all scenarios, and the performance ranking identified in Sec. 5.2.2 remains unchanged.

featuring relatively small query amounts (250 and 500) distributed over increasing numbers of communities. For instance, while the DA's default processing solutions are near-optimal for 500 queries distributed over six equal-sized communities, the performance gap against our incremental method substantially increases for varying community sizes, with default processing costs of up to 1.62.

For queries distributed equally among communities, the maximum community size is minimised. This renders dividing the corresponding MQO graphs into moderately-sized partitions less challenging, particularly for increasing amounts of communities that decrease individual community sizes. Such scenarios tend to be less challenging for the default DA processing method. In contrast, communities of varying size may feature large individual communities, which complicates partitioning when seeking to minimise information loss. While our DSS-based incremental method re-integrates initially discarded cost savings and maintains high optimisation performance, scenarios with potentially large communities cause a deterioration of the DA's default processing performance. Similarly to the unlikely scenario of an entirely uniform distribution of cost savings, communities cannot be assumed to be equal-sized in real query loads, and our method is likely to excel for real workloads.

5.2.4 Increasing Densities. We next discuss results for MQO scenarios whose community densities are sampled from intervals of increasing size, as depicted in Fig. 5. For our smallest interval, the maximum density is given by 0.25, while sampling from our largest interval can yield densities of up to 100%. For such large densities, the advantage of our incremental method over default processing tends to be most consistent and pronounced, and increases alongside growing amounts of queries. In contrast, the performance advantage varies more strongly in scenarios featuring lower densities: For the smallest density interval of $[0.05, 0.25]$, normalised default processing costs range between 1 and 1.4 for 750 queries, while the corresponding default processing costs for the $[0.05, 1]$ interval are much less varied, and consistently exceed our incremental costs more than twice. Finally, for small density intervals and low query numbers, default processing results sometimes yield slightly lower costs compared to our incremental approach.

The decreasing performance gap for smaller density intervals can be attributed to the overall weakening impact of communities resulting from smaller densities. With decreasing densities, MQO scenarios approximate uniform cases for which the advantage of our incremental processing method deteriorates. Scenarios with higher densities offer more structural properties that can be exploited.

Consider an outlier result for interval $[0.05, 0.75]$ and 750 queries, where our incremental method fails to provide an advantage over default processing for one scenario despite a relatively large density interval: Sampled densities happen to be very small for three of its four query communities (0.21 for one community, and 0.09 for two communities that feature 55% of all queries). While the fourth community has a larger density of 0.56, it features only 18 among the 750 queries, with negligible impact. While the advantage of incremental processing deteriorates in this specific scenario with particular characteristics (and in agreement with prior observations), it remains substantial for the bulk of problem scenarios.

5.3 Conventional QO Benchmarks

A comprehensive parameter sweep on all MQO input dimensions identifies scenarios where the advantage of our incremental processing method is most pronounced, but also shows some unfavourable cases. As the latter correspond to special cases not expected from typical problem loads (such as uniform savings distributions and balanced query communities of equal size), we have obtained a strong indication for the broader applicability of our approach. We next consider industrially realistic MQO scenarios generated based on traditional and established query optimisation (QO) benchmarks, including TPC-H [45], LDBC BI [1] and the join order benchmark [26].

5.3.1 Problem Generation. To generate problems from query optimisation benchmarks, we could consider all queries as an MQO query batch. However, this would limit problem sizes to the amount of queries contained in each benchmark. To reach dimensions sufficient for our large-scale method, we generate batches of queries by extrapolating from each set of benchmark queries, and derive cost savings based on overlapping relations between query pairs (details below). Hence, our extrapolation rests on the base relations featured by each benchmark: We generate queries by assigning relations in accordance with probabilities corresponding to their relative frequencies in the original benchmark queries. Hence, if a relation A is featured in 50% of all original benchmark queries, it is assigned to any generated query with probability 50%.

Next, we seek to assign cost savings in accordance with the commonalities featured by each pair of queries. To this end, we apply a simple metric that numerically captures the overlapping commonalities and conformance between two benchmark queries: We determine the conformance c_{q_1,q_2} between queries q_1 and q_2 as $c_{q_1,q_2} = Card_{ol}/Card_{ttl}$, where $Card_{ol}$ denotes the accumulated cardinality of all *overlapping* relations featured by both queries, and $Card_{ttl}$ denotes the accumulated *total* cardinality of all relations featured by *either* query. Thus, if q_1 and q_2 operate on entirely disjoint sets of relations ($Card_{ol} = 0$), their conformance will be assessed as $c_{q_1,q_2} = 0$, accurately reflecting that no cost savings opportunities arise for these queries. Based on this metric, we assign cost savings between query plans associated with benchmark queries q_i and q_j at probability c_{q_i,q_j}. Finally, the conformance values for all query pairs form a *conformance graph*, which we analyse w.r.t. potential community structures below. The remaining problem generation parameters are identical to those used for our parameter sweep. All generated problems can be found in our reproduction package [41].

5.3.2 Experimental Results. Fig. 6 depicts results for conventional benchmark scenarios. We consider DA default and incremental processing, as the DA's default processing method is closest to the latter. We omit parallel processing for the DA and default processing for SA, whose relatively weaker performance remains as in Fig. 3.

Firstly, we observe no pronounced changes in the relative ranking between algorithms. This includes, most importantly, our incremental processing method applied on the DA, whose performance advantage over the DA's default processing method remains substantial in nearly all scenarios, and consistently beats remaining competitors. This suggests, based on parameter sweep findings, that scenarios derived from conventional query optimisation benchmarks possess structural properties that can be exploited by our

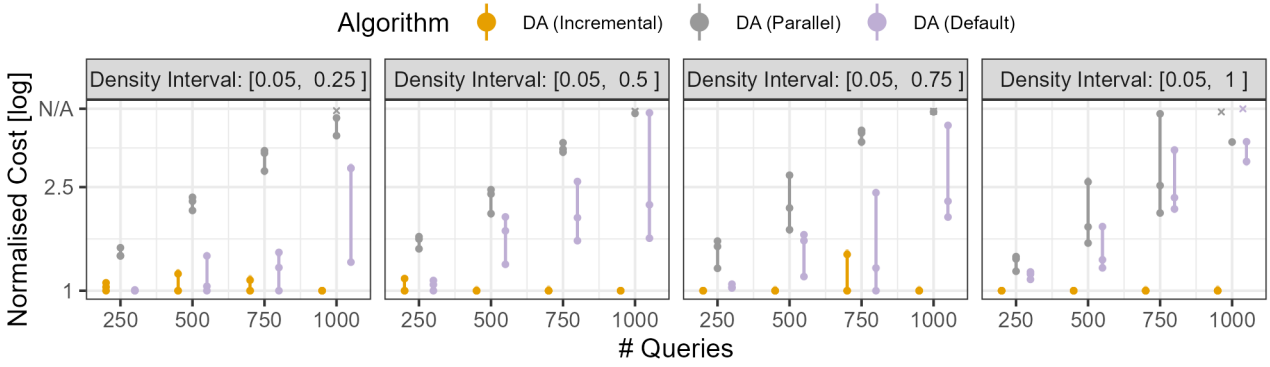

Figure 5: Normalised solution costs for problems with 30 PPQ and increasing numbers of queries, and four query communities of varying sizes. Community densities are sampled from increasing intervals (cf. Fig. 3 for details; costs ≥ 5 mapped to N/A).

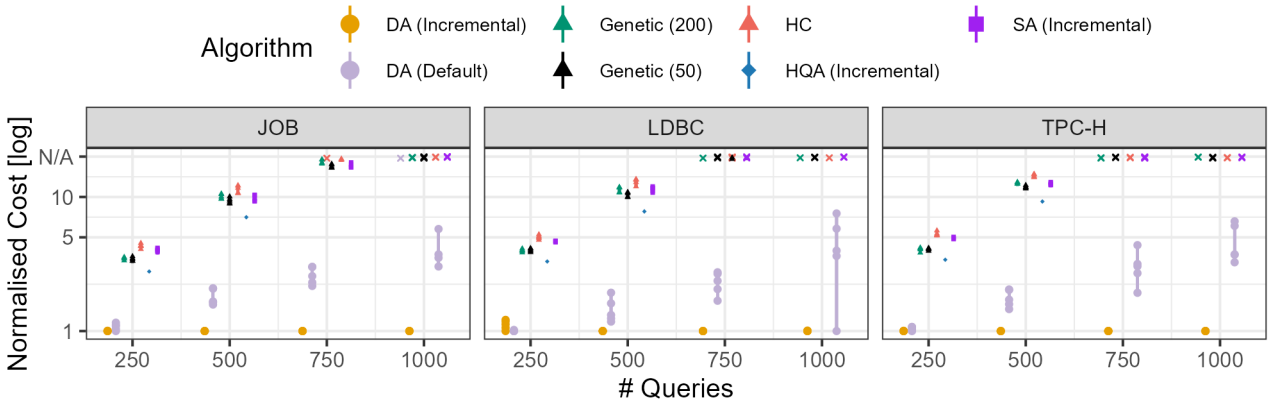

Figure 6: Normalised solution costs for problems with 30 PPQ and increasing numbers of queries (cf. Fig. 3 for details).

incremental processing method: Their query conformance graphs show distinct community structures, rather than purely uniform savings distributions. While JOB and LDBC scenarios feature two and four roughly equal-sized communities, TPC-H scenarios tend to feature one large (≈ 55% of all queries), one moderately-sized (≈ 28%) and one small community (≈ 17%). Performance results adhere to the advantage observed for our incremental method in scenarios with non-uniform savings distributions.

Closer inspection of the benchmark results reveals a wider variance in the performance gap between our incremental DA method and the DA's default processing for LDBC, compared to more consistent JOB results. The parameter sweep shows performance advantage of our incremental approach is most pronounced for large, distinct query communities, and tends to decrease as the number of communities increases, particularly with equal-sized communities. Both trends apply to the LDBC dataset: Compared to JOB scenarios with two communities, LDBC cases feature four equal-sized communities, which minimises the maximum size of individual communities. In contrast to JOB, the advantage of our incremental method is less pronounced for most LDBC scenarios, and even disappears for some LDBC problems below 500 queries.

For 1,000 queries, variance in LDBC solution quality further widens, suggesting an increasingly probabilistic partitioning heuristic is applied by the DA's default processing for large MQO scenarios. For one LDBC case and two TPC-H problems, the cost overhead of default DA processing over our incremental method exceeds the corresponding JOB overhead. This suggests that for 1,000 queries, even more numerous equal-sized communities feature individual sizes that are challenging for the DA's default processing, and a larger number of individual communities may constitute more challenging scenarios once problems feature very large query amounts.

The results for conventional benchmarks confirm our parameter sweep insights, and further indicate a broader applicability of our method.

5.4 Runtime Analysis

In Fig. 7, we analyse runtimes for increasing amounts of queries and savings densities up to 0.8. Our analysis rests on our annealing-based methods and varying processing methods. We do not consider the genetic algorithm and hill climbing, as they yield intermediate results after each optimisation step until timeout, rather than once after termination (solution quality remains sub-par even after 300s).

For our annealing methods, we compare runtimes of up to 180s. All scenarios show substantial runtime overheads for default SA

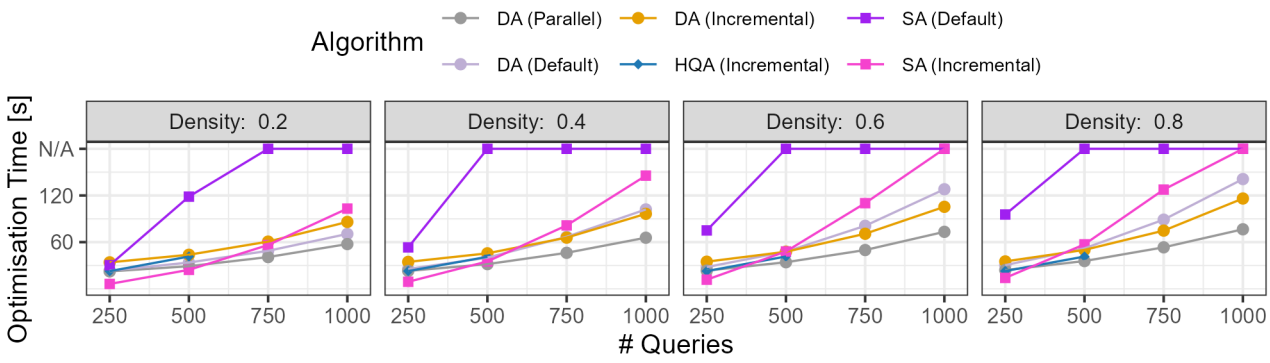

Figure 7: Optimisation times for various MQO algorithms, yielded for MQO problems featuring increasing numbers of queries, 30 PPQ, as well as increasing savings densities. Cases where no result is obtained before 180s are labeled by N/A.

compared to DA and HQA, which further demonstrates the performance advantage of their special-purpose HW. However, when augmented with our incremental processing method, SA times are significantly reduced, showing that SA runtime benefits greatly from the problem size reduction. For 250 queries, incremental SA runtime is lower compared to competitors. Yet, SA performance stagnates with growing size, and its runtime significantly exceeds the remaining competitors for 1,000 queries and larger densities.

Incremental HQA yields similar runtimes as default and incremental DA processing for a savings density of 0.2. For increasing densities, HQA runtime scales more gracefully, as HQA terminates after 40s of optimisation time compared to 50s required by default and incremental DA, for 500 queries. This suggests the density of MQO problems to be particularly influential on the DA's runtime scalability.

Analysing our varying processing methods confirms this observation: For a density of 0.2 and 1,000 queries, the runtime of our incremental strategy tends to be slightly higher (86s) than default processing times (70s). Yet, the ranking reverses as the density increases, and for a density of 0.8 and 1,000 queries, default runtime (141s) distinctly exceeds that of our incremental method (116s). To explain this change in runtime behavior, we may consider two opposing runtime properties: Firstly, the computational runtime overhead induced by our partitioning and DSS steps, and secondly, the reduced annealing runtime resulting from the problem reduction. As the density increases, the runtime benefit of the problem size reduction outweighs the overhead of partitioning and DSS, rendering our incremental method more efficient for increasing densities. Even for smaller densities with no runtime advantage, the runtime overhead of our incremental method remains moderate when accounting for the increase in solution quality discussed in detail above. Finally, allowing for a parallel processing of partial problems, the parallel processing strategy naturally yields distinctly lower runtimes compared to other methods. However, our analysis above has shown its substantial disadvantage in solution quality, compared to our incremental method that yields normalised optimal solutions for the bulk of all considered MQO scenarios.

6 Related Work

MQO is a classical optimisation problem in the DB domain [42]. It broadly involves two fundamental phases: (a) identifying common sub-expressions between queries and generating execution plans [9, 25], and (b) identifying ideal plan selections exploiting cost saving opportunities based on identified common sub-expressions. The method proposed in this paper addresses category (b), and hence relates to approaches seeking ideal plan selections based on previously determined plans and sub-expressions.

Our method builds on the quantum approach by Trummer and Koch [47], which solves small classes of MQO problems on quantum annealers, yet is severely limited by their capacity constraints (recall Fig. 1). In contrast, our method achieves scalability beyond such limits by applying partitioning and incremental optimisation strategies. We provide a novel alternative to established MQO approaches, which include methods based on the A-* algorithm [10, 42, 43] (suited to determine optimal solutions for small problems) or heuristics seeking to determine near-optimal solutions for larger problems (such as genetic algorithms [4], hill climbing [12], and integer linear programming [11]). In comparison, our method shows substantially increased robustness against extremely large solution spaces.

Further, our approach aligns with recent trends in the DB domain that seek to address the limitations of conventional general-purpose systems by shifting towards highly optimised special-purpose accelerators like FPGA modules [30–32], or GPUs [17].

Recent work considers the potential of quantum computers, and involves approaches rendering problems compatible with quantum devices [3], as well as methods boosting the efficiency of existing quantum algorithms [19, 34, 36]. The prevailing quantum paradigms include quantum machine learning, as well as QUBO-based quantum optimisation for various problems in the DB domain, such as transaction scheduling [6–8, 21] and join order optimisation [18, 38, 40, 48]. For the latter, previous work assessed the potential of hardware-software co-design methods to optimise performance [35, 38, 49]. Finally, Trummer and Koch derived the QUBO-encoding for MQO, solving small problem classes on a quantum annealing device [47]. The method was further evaluated on gate-based quantum systems [14, 37]. Yet, given the limitations of current quantum systems, none of these approaches have targeted large-scale problems, which have been the focus of this paper.

In contrast, quantum-inspired devices are robust for larger problems. Matsubara *et al.* [28] observe speedups by orders of magnitude on the DA for a variety of combinatorial optimisation problems. Similar results are known in other areas [13, 22, 24, 44]. Within the DB domain, Schönberger *et al.* [39] have used the DA for join order optimisation. However, all of these approaches are limited by HW capacity restrictions. In contrast, our incremental annealing method tailored to MQO overcomes these limits, and has been empirically shown to outperform general partitioning approaches supported by various quantum(-inspired) annealers.

7 Discussion and Conclusion

Large-scale database workloads require optimisation across a range of critical dimensions. This includes maximising machine utilisation and throughput, minimising response latencies, and optimising energy efficiency. Mounting challenges on general-purpose hardware prompt a shift towards special-purpose HW accelerators.

Efficient multi-query optimisation holds considerable promise: While implementations on conventional hardware have reached scalability limits, our novel approach combines quantum-inspired hardware accelerators with strategies for complexity reduction and decomposition, as well as means of improving result quality. We have successfully overcome hardware constraints and improved over both, conventional MQO methodologies and alternative partitioning strategies provided by hardware vendors. We observe robust performance even for exceptionally large MQO scenarios involving up to 1,000 queries — problem scales known to be intractable for competing techniques.

However, we did not only achieve significant performance improvements over state-of-the-art methods across multiple dimensions, but also believe our framework lays the groundwork for potential extensions and generalisations to other DBMS challenges. For instance, we observe structural similarities between the traditional DB problem of join ordering (JO) and MQO, as both feature graph-based representations. This suggests our approach may be adaptable to JO, where nodes and edges correspond to relations and join predicates: Similarly to our MQO partitioning graph, we may reduce JO node amounts by merging node sets based on criteria such as interconnectivity, and apply graph partitioning to decompose the problem while minimising the loss of information.

Finally, we establish a path towards mature quantum systems that are expected to surpass the capabilities of surrogate technologies. While the commercial realisation of such systems likely remains years away, our work provides a bridge to this promising future, positioning it as an essential contribution to the field.

Acknowledgments

MS and WM were supported by the German Federal Ministry of Education and Research (BMBF), funding program "Quantum Technologies—from Basic Research to Market", grant 13N16092. WM acknowledges support by the High-Tech Agenda Bavaria.

References

[1] Renzo Angles, Peter Boncz, Josep-Lluis Larriba-Pey, Irini Fundulaki, Thomas Neumann, Orri Erling, Peter Neubauer, Norbert Martínez-Bazan, Venelin Kotsev, and Ioan Toma. 2014. The Linked Data Benchmark Council: A graph and RDF industry benchmarking effort. *ACM SIGMOD Record* 43 (05 2014), 27–31. doi:10.1145/2627692.2627697

[2] Maliheh Aramon, Gili Rosenberg, Elisabetta Valiante, Toshiyuki Miyazawa, Hirotaka Tamura, and Helmut G. Katzgraber. 2019. Physics-Inspired Optimization for Quadratic Unconstrained Problems Using a Digital Annealer. *Frontiers in Physics* 7 (2019). doi:10.3389/fphy.2019.00048

[3] Andreas Bayerstadler, Guillaume Becquin, Julia Binder, Thierry Botter, Hans Ehm, Thomas Ehmer, Marvin Erdmann, Norbert Gaus, Philipp Harbach, Maximilian Hess, Johannes Klepsch, Martin Leib, Sebastian Luber, Andre Luckow, Maximilian Mansky, Wolfgang Mauerer, Florian Neukart, Christoph Niedermeier, Lilly Palackal, Ruben Pfeiffer, Carsten Polenz, Johanna Sepulveda, Tammo Sievers, Brian Standen, Michael Streif, Thomas Strohm, Clemens Utschig-Utschig, Daniel Volz, Horst Weiss, and Fabian Winter. 2021. Industry Quantum Computing Applications. *EPJ Quantum Technology* 8, 1 (11 2021). doi:10.1140/epjqt/s40507-021-00114-x

[4] Murat Ali Bayir, Ismail H. Toroslu, and Ahmet Cosar. 2007. Genetic Algorithm for the Multiple-Query Optimization Problem. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 37, 1 (2007), 147–153. doi:10.1109/TSMCC.2006.876060

[5] Zhengbing Bian, Fabián A. Chudak, William G. Macready, and Geordie Rose. 2010. The Ising model : Teaching an Old Problem New Tricks. https://www.dwavesys.com/media/vbklsvbh/weightedmaxsat_v2.pdf

[6] Tim Bittner and Sven Groppe. 2020. Avoiding Blocking by Scheduling Transactions Using Quantum Annealing. In *Proceedings of the 24th Symposium on International Database Engineering & Applications* (Seoul, Republic of Korea) *(IDEAS '20)*. Association for Computing Machinery, New York, NY, USA, Article 21, 10 pages. doi:10.1145/3410566.3410593

[7] 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.

[8] 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 (jul 2023), 2344–2353. doi:10.14778/3598581.3598603

[9] F-CF Chen and Margaret H Dunham. 1998. Common subexpression processing in multiple-query processing. *IEEE Transactions on Knowledge and Data Engineering* 10, 3 (1998), 493–499.

[10] Ahmet Cosar, Ee-Peng Lim, and Jaideep Srivastava. 1993. Multiple Query Optimization with Depth-First Branch-and-Bound and Dynamic Query Ordering. In *Proceedings of the Second International Conference on Information and Knowledge Management* (Washington, D.C., USA) *(CIKM '93)*. Association for Computing Machinery, New York, NY, USA, 433–438. doi:10.1145/170088.170181

[11] Tansel Dokeroglu, Murat Ali Bayır, and Ahmet Cosar. 2014. Integer Linear Programming Solution for the Multiple Query Optimization Problem. In *Information Sciences and Systems 2014*, Tadeusz Czachórski, Erol Gelenbe, and Ricardo Lent (Eds.). Springer International Publishing, Cham, 51–60.

[12] Tansel Dokeroglu, Murat Ali Bayir, and Ahmet Cosar. 2015. Robust Heuristic Algorithms for Exploiting the Common Tasks of Relational Cloud Database Queries. *Appl. Soft Comput.* 30, C (may 2015), 72–82. doi:10.1016/j.asoc.2015.01.026

[13] Sebastian Engel, Christian Münch, Fritz Schinkel, Oliver Holschke, Marc Geitz, and Timmy Schüller. 2022. Segment Routing with Digital Annealing. In *NOMS 2022-2022 IEEE/IFIP Network Operations and Management Symposium*. 1–9. doi:10.1109/NOMS54207.2022.9789782

[14] Tobias Fankhauser, Marc E. Solèr, Rudolf Marcel Füchslin, and Kurt Stockinger. 2023. Multiple Query Optimization Using a Gate-Based Quantum Computer. *IEEE Access* 11 (2023), 114031–114043. doi:10.1109/ACCESS.2023.3324253

[15] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. 2014. A Quantum Approximate Optimization Algorithm. arXiv:1411.4028 [quant-ph]

[16] Edward Farhi, Jeffrey Goldstone, Sam Gutmann, and Michael Sipser. 2000. Quantum Computation by Adiabatic Evolution. arXiv:quant-ph/0001106 [quant-ph]

[17] Johannes Fett, Annett Ungethüm, Dirk Habich, and Wolfgang Lehner. 2021. The Case for SIMDified Analytical Query Processing on GPUs. In *Proceedings of the 17th International Workshop on Data Management on New Hardware (DaMoN 2021)* (Virtual Event, China) *(DAMON'21)*. Association for Computing Machinery, New York, NY, USA, Article 14, 5 pages. doi:10.1145/3465998.3466015

[18] Maja Franz, Tobias Winker, Sven Groppe, and Wolfgang Mauerer. 2024. Hype or Heuristic? Quantum Reinforcement Learning for Join Order Optimisation. In *Proceedings of the IEEE International Conference on Quantum Computing and Engineering*. https://arxiv.org/abs/2405.07770

[19] Martin Gogeißl, Hila Safi, and Wolfgang Mauerer. 2024. Quantum Data Encoding Patterns and their Consequences. In *Proceedings of the Workshop on Quantum Computing and Quantum-Inspired Technology for Data-Intensive Systems and Applications (QDSM '24)*. doi:10.1145/3665225.3665446

[20] Felix Greiwe, Tom Krüger, and Wolfgang Mauerer. 2023. Effects of Imperfections on Quantum Algorithms: A Software Engineering Perspective. In *2023 IEEE International Conference on Quantum Software (QSW)*. 31–42. doi:10.1109/QSW59989.

2023.00014
[21] Sven Groppe and Jinghua Groppe. 2021. Optimizing Transaction Schedules on Universal Quantum Computers via Code Generation for Grover's Search Algorithm. In *25th International Database Engineering & Applications Symposium* (Montreal, QC, Canada) *(IDEAS 2021)*. Association for Computing Machinery, New York, NY, USA, 149–156. doi:10.1145/3472163.3472164
[22] Seo Woo Hong, Pierre Miasnikof, Roy Kwon, and Yuri Lawryshyn. 2021. Market Graph Clustering via QUBO and Digital Annealing. *Journal of Risk and Financial Management* 14, 1 (2021). doi:10.3390/jrfm14010034
[23] D-Wave Systems Inc. 2020. *D-Wave Hybrid Solver Service: An Overview*. Technical Report 14-1039A-B. D-Wave Systems Inc. https://www.dwavesys.com/media/4bnpi53x/14-1039a-b_d-wave_hybrid_solver_service_an_overview.pdf
[24] Tomoki Inoue, Tsubasa Ikami, Yasuhiro Egami, Hiroki Nagai, Yasuo Naganuma, Koichi Kimura, and Yu Matsuda. 2023. Data-driven optimal sensor placement for high-dimensional system using annealing machine. *Mechanical Systems and Signal Processing* 188 (2023), 109957. doi:10.1016/j.ymssp.2022.109957
[25] Matthias Jarke. 1985. Common subexpression isolation in multiple query optimization. In *Query Processing in Database Systems*. Springer, 191–205.
[26] Viktor Leis, Bernhard Radke, Andrey Gubichev, Atanas Mirchev, Peter Boncz, Alfons Kemper, and Thomas Neumann. 2018. Query optimization through the looking glass, and what we found running the join order benchmark. *The VLDB Journal* 27 (2018), 643–668.
[27] Andrew Lucas. 2014. Ising Formulations of Many NP Problems. *Frontiers in Physics* 2 (2014), 5. doi:10.3389/fphy.2014.00005
[28] Satoshi Matsubara, Motomu Takatsu, Toshiyuki Miyazawa, Takayuki Shibasaki, Yasuhiro Watanabe, Kazuya Takemoto, and Hirotaka Tamura. 2020. Digital Annealer for High-Speed Solving of Combinatorial optimization Problems and Its Applications. In *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*. 667–672. doi:10.1109/ASP-DAC47756.2020.9045100
[29] Klaus Meffert and Neil Rotstan. 2007. Jgap: Java genetic algorithms package. (2007). https://homepages.ecs.vuw.ac.nz/~lensenandr/jgap/documentation/
[30] Adrian Michalke, Philipp M. Grulich, Clemens Lutz, Steffen Zeuch, and Volker Markl. 2021. An Energy-Efficient Stream Join for the Internet of Things. In *Proceedings of the 17th International Workshop on Data Management on New Hardware (DaMoN 2021)* (Virtual Event, China) *(DAMON'21)*. Association for Computing Machinery, New York, NY, USA, Article 8, 6 pages. doi:10.1145/3465998.3466005
[31] Mehdi Moghaddamfar, Christian Färber, Wolfgang Lehner, and Akash Kumar. 2023. KeRRaS: Sort-Based Database Query Processing on Wide Tables Using FPGAs. In *Proceedings of the 19th International Workshop on Data Management on New Hardware*. 1–9.
[32] Mehdi Moghaddamfar, Christian Färber, Wolfgang Lehner, Norman May, and Akash Kumar. 2021. Resource-Efficient Database Query Processing on FPGAs. In *Proceedings of the 17th International Workshop on Data Management on New Hardware (DaMoN 2021)* (Virtual Event, China) *(DAMON'21)*. Association for Computing Machinery, New York, NY, USA, Article 4, 8 pages. doi:10.1145/3465998.3466006
[33] NEC Corporation. 2022. *Vector Annealing*. https://www.nec.com/en/global/solutions/hpc/event/supercomputing/pdf/4_SC22_NEC_Aurora_Forum.pdf
[34] Maniraman Periyasamy, Axel Plinge, Christopher Mutschler, Daniel D. Scherer, and Wolfgang Mauerer. 2024. Guided-SPSA: Simultaneous Perturbation Stochastic Approximation assisted by the Parameter Shift Rule. In *Proceedings of the IEEE International Conference on Quantum Computing and Engineering*. doi:10.1109/QCE60285.2024.00177
[35] 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)*. 104–115. doi:10.1109/QSW59989.2023.00022
[36] Lukas Schmidbauer, Karen Wintersperger, Elisabeth Lobe, and Wolfgang Mauerer. 2024. Polynomial Reduction Methods and their Impact on QAOA Circuits. In *Proceedings of the IEEE International Conference on Quantum Software (QSW)*. doi:10.1109/QSW62656.2024.00018
[37] Manuel Schönberger. 2022. Applicability of Quantum Computing on Database Query Optimization. In *Proceedings of the 2022 International Conference on Management of Data* (Philadelphia, PA, USA) *(SIGMOD '22)*. Association for Computing Machinery, New York, NY, USA, 2512–2514. doi:10.1145/3514221.3520257
[38] 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 (may 2023), 27 pages. doi:10.1145/3588946
[39] Manuel Schönberger, Immanuel Trummer, and Wolfgang Mauerer. 2023. Quantum-Inspired Digital Annealing for Join Ordering. In *Proceedings of the VLDB Endowment*, Vol. 17. doi:10.14778/3632093.3632112
[40] Manuel Schönberger, Immanuel Trummer, and Wolfgang Mauerer. 2023. Quantum Optimisation of General Join Trees. In *Joint Workshops at 49th International Conference on Very Large Data Bases (VLDBW'23) — International Workshop on Quantum Data Science and Management (QDSM '23)*.
[41] Manuel Schönberger, Immanuel Trummer, and Wolfgang Mauerer. 2025. Github reproduction repository. Retrieved May 26, 2025 from https://github.com/lfd/sigmod26
[42] Timos K. Sellis. 1988. Multiple-Query Optimization. *ACM Trans. Database Syst.* 13, 1 (mar 1988), 23–52. doi:10.1145/42201.42203
[43] Kyuseok Shim, Timos Sellis, and Dana Nau. 1994. Improvements on a Heuristic Algorithm for Multiple-Query Optimization. *Data Knowl. Eng.* 12, 2 (mar 1994), 197–222. doi:10.1016/0169-023X(94)90014-0
[44] Masahiko Sugimura, Mikinori Kobayashi, Hidetoshi Matsumura, Xi Wang, and Paparao Palacharla. 2022. Accelerate Optical Network Modernization through Quantum-inspired Digital Annealing. In *2022 European Conference on Optical Communication (ECOC)*. 1–4.
[45] Transaction Processing Performance Council . 2023. TPC Benchmark H. Retrieved November 10, 2023 from https://www.tpc.org/
[46] Immanuel Trummer. 2022. Multiple Query Optimization on the D-Wave Quantum Annealer. Retrieved December 6, 2023 from https://github.com/itrummer/quantumdb
[47] Immanuel Trummer and Christoph Koch. 2016. Multiple Query Optimization on the D-Wave 2X Adiabatic Quantum Computer. *Proc. VLDB Endow.* 9, 9 (may 2016), 648–659. doi:10.14778/2947618.2947621
[48] 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 International Conference on Management of Data* (Seattle, WA, USA) *(SIGMOD/PODS '23)*. Association for Computing Machinery, New York, NY, USA, 45–52. doi:10.1145/3555041.3589404
[49] Karen Wintersperger, Hila Safi, and Wolfgang Mauerer. 2022. QPU-System Co-Design for Quantum HPC Accelerators. In *Proceedings of the 35th GI/ITG International Conference on the Architecture of Computing Systems*, Martin Schulz, Carsten Trinitis, Nikela Papadopoulou, and Thilo" Pionteck (Eds.). Gesellschaft für Informatik, 100–114. doi:10.1007/978-3-031-21867-5_7