Index Advisors on Quantum Platforms

Manish Kesarwani Jayant Haritsa

Technical Report
TR-2024-01
(June 2024)

Database Systems Lab
Indian Institute of Science
Bangalore 560012, India
http://dsl.cds.iisc.ac.in

1 Introduction

Given an SQL query workload, the creation of appropriate column indexes has been a standard database technique for materially reducing the workload's execution time. In the early days, these indexes were manually selected by DBAs. However, contemporary engines feature automated Index Advisors (IA) that identify good configurations while adhering to storage budgets; for instance, IBM's DB2 Index Advisor [55] and Microsoft's AutoAdmin [28].

Given their demonstrated performance impact, it is no surprise that IA design has been an active area of research over the past three decades in both academia and industry (e.g. [15, 21, 22, 28, 25, 26, 30, 31, 52, 55, 61, 27, 39, 51]), with even machine learning-based techniques [42, 43, 17, 48] appearing in recent times. However, searching for an optimal index configuration inherently entails exploring a combinatorially large search space. Therefore, current advisors typically rely on heuristic strategies, which can result in suboptimal index configurations. For instance, DB2 IA reduces the index selection problem to an instance of a 0-1 Knapsack Problem [50], and then invokes a greedy heuristic-based solver to recommend the index configuration. The heuristic is essentially "ROI" (return on investment) – the time benefit provided by the index normalized to its storage footprint.

Problem Instance

Index (I)	Columns in the Index	Time Benefit (V)	Storage Cost (W)
i_0	[L_DISCOUNT, L_SHIPDATE, L_QUANTITY, L_EXTENDEDPRICE]	165811	266
i_1	[L_SHIPDATE, L_DISCOUNT, L_EXTENDEDPRICE, L_PARTKEY]	178871	232
i_2	[P_CONTAINER, P_BRAND, P_PARTKEY]	1213770	8
i_3	[L_PARTKEY, L_QUANTITY]	1213770	132
i_4	[L_PARTKEY, L_QUANTITY, L_EXTENDEDPRICE]	1213770	199
i_5	[C_ACCTBAL, C_CUSTKEY, C_PHONE]	44370	2
i_6	[O_CUSTKEY]	44370	9

***Storage Capacity*: 140**

CDB Recommendation	***Optimal Configuration***
Heuristic Index Set: $\{i_2, i_5, i_6\}$	*Optimal Index Set:* $\{i_2, i_3\}$
Heuristic Benefit: 1302510	*Optimal Benefit: 2427540*

Figure 1: Suboptimality of Heuristic IA

Consider an SQL workload comprising TPC-H [9] queries Q6, Q14, Q22, and two instances of Q17 over a 1GB TPC-H database. Given this setup, the index selection problem instance generated by a popular commercial database (CDB) engine is shown in Figure 1. The problem instance comprises seven candidate indexes, their expected time benefit wrt query response time, and storage cost overhead. Now, for a storage budget of 140, CDB recommends a sub-optimal index configuration $\{i_2, i_5, i_6\}$ with a total benefit of 1302510, while the optimal configuration comprises indexes $\{i_2, i_3\}$ with a benefit of 2427540. In this scenario, it is evident that around 50% of the available index benefit is lost due to a sub-optimal choice.

As highlighted in [21], a variety of heuristics have been proposed for classical IA tools in an attempt to bridge this gap to optimality – however, there is no guaranteed improvement. Therefore, we take

a radically different approach here: specifically, we ask the question "Is it feasible to utilize the raw computational power promised by *quantum platforms* to find better, perhaps even optimal, solutions?". And the good news, as explained in the remainder of this paper, is that it indeed appears viable, through careful algorithmic design, to concurrently achieve excellent quality and practical efficiency. As a case in point, the optimal configuration for the ~ 50 candidate indexes constructable on a TPC-H database, can be processed (as per our estimated projection in Section 6.7.4) in a few hours on a quantum computer, whereas an exhaustive search would take a couple of *months*.

Our study is motivated by the growing interest in early-stage quantum computing with 100,000-qubit machines projected within the coming decade [3]. Even within the database community, quantum computing has begun to attract attention – similar, albeit unrelated, studies to ours have recently been carried out for join-order optimization [53, 46, 58] and transaction scheduling [18, 19, 36], with promising outcomes. Finally, a call for quantum implementation of Index Advisors was explicitly advocated in recent database vision papers [38, 59].

Quantum Modeling Dimensions

A variety of design choices and challenges arise in porting IA to the quantum domain. First, there are alternative computing models – *quantum annealing*, which is energy-minimization-based, and *quantum circuit*, which is gate-based, similar to classical circuits. Over the past decade, the latter has gained prominence as it provides a greater degree of design flexibility [53], and we therefore focus on this choice in our work.

Second, we could ask whether the IA design should be purely quantum or a hybrid that synergistically leverages classical and quantum computing. We have chosen the latter to (a) facilitate easy integration with contemporary DBMS engines and (b) minimize the quantum circuit complexity to address only the hard computational problems.

Third, quantum computers work in *probabilistic* space – this means that individual results may have errors or even violate mandatory constraints. We therefore need to devise schemes to eliminate, with at least high confidence, these inherent problems of quantum computation.

Fourth, whether IA should be modeled as a *optimization* problem or as a *search* formulation. Due to the disparate trade-offs between their solution quality and computational effort, we design and evaluate both options. For the former, index selection is modeled as a Quadratic Unconstrained Binary Optimization problem, and solved using the Quantum Approximate Optimization Algorithm (QAOA) [33]. This approach requires $O(\log(L))$ computations, where L is the total number of candidate configurations, and the recommended configurations are significantly better compared to classical heuristics. On the other hand, for the latter, index selection is modeled as a fully enumerative search over the exponential configuration space, and solved using the seminal Grover Search algorithm [37]. This approach identifies, with high confidence, the *optimal* index configuration incurring $O(\sqrt{L})$ computations.

Finally, modeling database applications on quantum platforms poses tricky implementation issues. For instance, with Grover Search, we have to devise an efficient data loading scheme that can scale with the size of the problem and also facilitate subsequent computations. Our approach diverges from the conventional method of loading data via *basis states* – instead, we load the data in the *phase* of the qubits. This unconventional strategy allows for more efficient computation and paves the way for further innovations (detailed in Section 5.1). We also design an efficient quantum *oracle* that is able to identify qualifying configurations. The creation of such an oracle is a complex task, as it requires performing computations while the data is phase-resident in a quantum superposition state.

The QIA System

The overall architecture of our hybrid Quantum-Classical Index Advisor (QIA) framework is shown in Figure 2. Given a database environment, the system takes the SQL workload and storage budget as input and outputs a recommended index configuration.

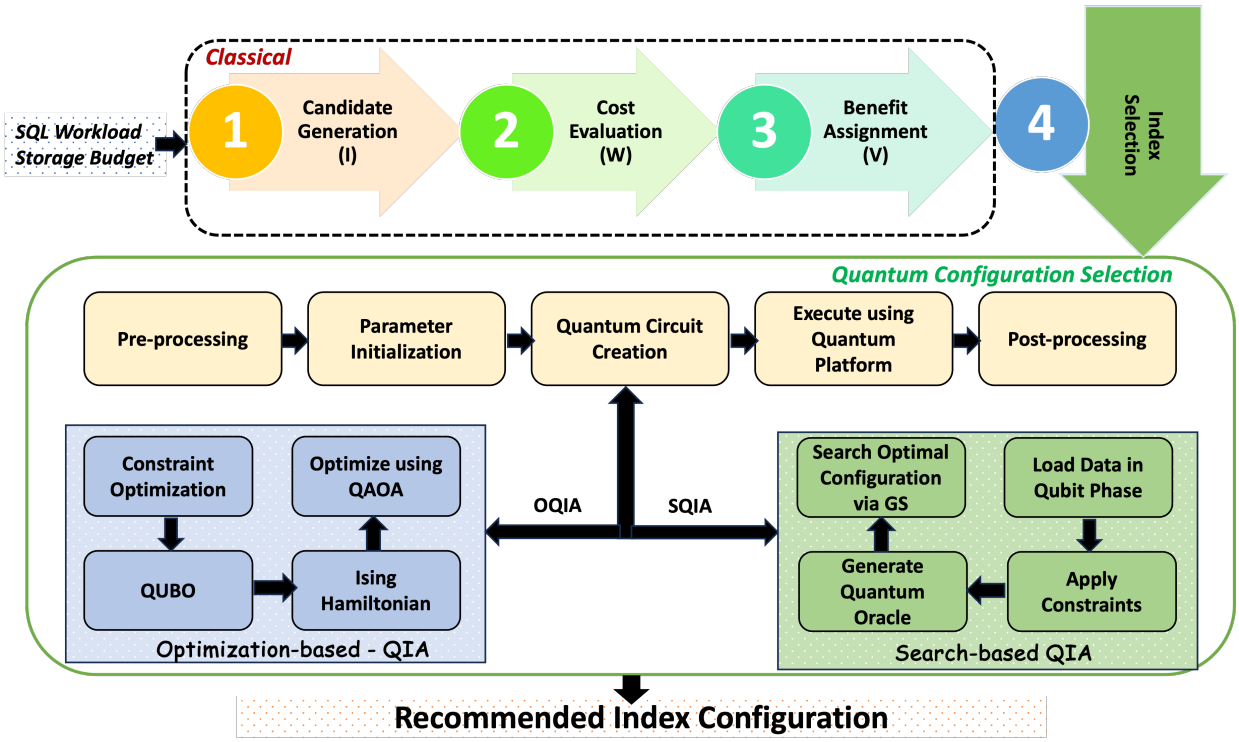


Figure 2: Quantum Index Advisor (QIA) Architecture

The first three steps – candidate configuration generation, followed by computing the storage costs and time benefits of these configurations – are carried out in the classical world, whereas the final computationally intensive index selection step is hosted on the quantum platform. That is, the existing index selection heuristic algorithm is replaced by a quantum substitute, while maintaining the same classical interface, in a pluggable manner. Now, based on the user choice, either OQIA (Optimization-based QIA) or SQIA (Search-based QIA) is invoked to recommend the index configuration. With SQIA, the user provides an additional parameter, δ, which is the desired probability of obtaining the optimal solution. While the OQIA porting is an amalgamation of known techniques, SQIA represents, to our knowledge, the first application of quantum search to the index selection problem implementable through standard quantum gates.

Evaluation

We have designed quantum circuits for the OQIA and SQIA algorithms and implemented them using the Qiskit SDK [16]. They have been evaluated on a **32-qubit** noiseless quantum simulator, and on a **127-qubit** IBM Eagle circuit processor. Given the current limited capacity of quantum platforms, we are perforce only able to model modest instances of the IA problem, which we have evaluated on the TPC-H environment.

However, interestingly, we show that even for these modest instances, using QIA, substantive quality improvements, approaching optimality, are obtained compared to the heuristic approach implemented

in a commercial DBMS engine. As a case in point, for the problem instance depicted in Figure 1, the quality of the CDB recommendation is 0.54 of the optimal (determined by exhaustive search). In contrast, OQIA delivers a 0.76 solution, while SQIA recommends the optimal with probability 0.9.

Moreover, these good results are obtained while incurring substantially less computational effort than the exhaustive search. Specifically, with OQIA, the computational overheads are 0.23 relative to exhaustive search, while SQIA is 0.77. Finally, we make the case that for large problem instances, our techniques effectively scale the qubit requirements *linearly* with problem size, an essential feature from a long-term feasibility perspective.

Index Advisor Framework

To put the Index Advisor framework into perspective, Figure 3 shows a high-level characterization of the QIA techniques, contrasted to the greedy and exhaustive approaches. The dimensions are the (normalized) index configuration quality, the computational efficiency in identifying these configurations, and the probabilistic distribution of the quality. On the efficiency axis, $L = 2^I$, where I is the number of indexes.

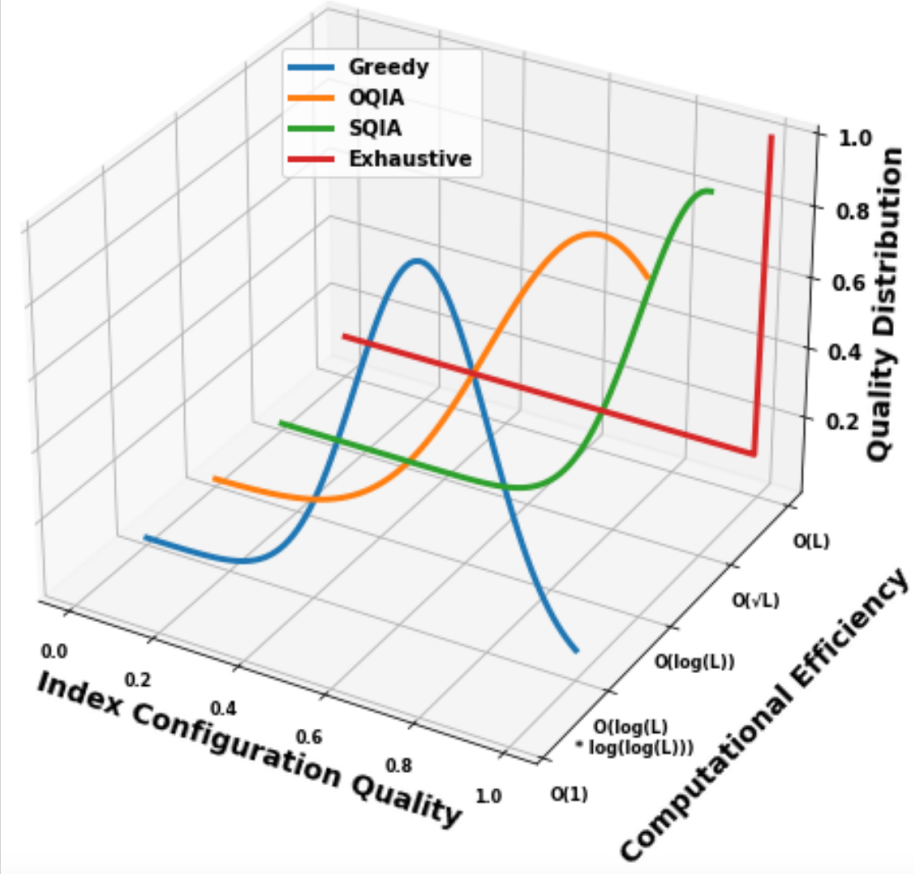


Figure 3: Quality-Efficiency Characterization

We see here that OQIA consistently delivers high-quality index configurations while performing fewer computations than Greedy. Further, by making a modest additional computational investment, SQIA could be instead used to obtain optimal configurations with a high probability. But we hasten to add that the computations performed by Greedy are simpler, and hence it may be empirically faster for contemporary index sizes. Therefore, a variety of quality-efficiency trade-offs are available, and in a

complete deployment, a sentinel module would be required to make the appropriate choice. We plan to explore this aspect in our future work.

Contributions

To summarize, our contributions are the following:

1. Proposed the first hybrid IA architecture that harnesses classical computing and gate-based quantum technology in a pluggable manner for database engines.

2. Logical design and circuit representations for the OQIA and SQIA approaches. The SQIA design, in particular, represents an original application of quantum search to the index selection problem, leveraging a phase-resident approach to data storage while only employing standard quantum gates. Further, OQIA and SQIA provide different tradeoffs between configuration quality and computational efficiency.

3. A pilot implementation and evaluation of the proposed IA approaches on both (noiseless) quantum simulators and (noisy) quantum circuit processors. The evaluations show that substantively improved index configurations, by a multiplicative factor of 1.5 to 2 and approaching optimality, are achievable through quantum technology. We also observe that OQIA is more robust to noise than SQIA.

To our knowledge, this study represents the first investigation of quantum computing to the IA problem. For this initial analysis, we restrict our attention to the computationally expensive index selection step in the IA pipeline. We intend to explore quantum implementation of other pipeline components in our future work.

The rest of the paper is organized as follows: A brief background of quantum data processing is given in Section 2. The formal problem framework is detailed in Section 3. Then, in Sections 4 and 5, we present the OQIA and SQIA algorithms, respectively, and their performance evaluation is profiled in Section 6. Related work is reviewed in Section 7. Finally, our conclusions and future research avenues are highlighted in Section 8.

2 Quantum Background

Quantum computation brings to bear on information processing, the fundamental phenomena of quantum mechanics, such as *superposition*, *interference*, *entanglement*, *reversible computation*, and *irreversible measurements*. A comprehensive review of quantum computation is provided in [47]. Here, we briefly review the basic building blocks used in QIA.

Quantum computation is built upon the *quantum bit* or *qubit*. The possible states for a qubit are $|0\rangle$ and $|1\rangle$ (in Dirac notation), which correspond to the states 0 and 1 of a classical bit. But unlike a classical bit, a qubit can be in a state $|\psi\rangle$ which is a linear combination, or superposition, of the $|0\rangle$ and $|1\rangle$ states:

$$|\psi\rangle = \alpha|0\rangle + e^{i\gamma}\beta|1\rangle \tag{1}$$

where α and β are complex numbers such that $|\alpha|^2 + |\beta|^2 = 1$ and $\gamma \in [0, 2\pi)$ is the quantum phase (angle of rotation around the Z-axis). When a qubit is measured, we get either 0 with probability $|\alpha|^2$, or 1 with probability $|\beta|^2$. Note that the phase γ *gets lost* in the measurement and, furthermore, the measurement operation is irreversible, since it destroys the quantum superposition state and outputs classical bits.

2.1 Quantum Gates

A quantum algorithm generates a quantum circuit comprising elementary quantum gates wired together to accomplish a task. The quantum gates used in QIA are summarized below:

NOT Gate (X): Operates on a single qubit and is quantum equivalent of the classical `NOT` gate. It takes a state $\alpha|0\rangle + \beta|1\rangle$ as input and flips the amplitudes of $|0\rangle$ and $|1\rangle$, producing the state $\beta|0\rangle + \alpha|1\rangle$.

Hadamard Gate (H): Operates on a single qubit and produces an *equal* superposition of the $|0\rangle$ and $|1\rangle$ states. That is, it turns $|0\rangle$ into $|+\rangle = (|0\rangle + |1\rangle)/\sqrt{2}$, and $|1\rangle$ into $|-\rangle = (|0\rangle - |1\rangle)/\sqrt{2}$.

Phase Gate (P): Takes an angle ϕ as input and rotates a qubit about the Z-axis, mapping: $|0\rangle \rightarrow |0\rangle$ and $|1\rangle \rightarrow e^{i\phi}|1\rangle$.

Controlled-Phase Gate (CP): Takes an angle ϕ as input and operates on a pair of qubits: a *control* qubit and a *target* qubit. The **P** gate is applied to the target qubit conditional on the state of the control qubit.

Multi-Controlled Toffoli Gate (MCT): Operates on a set of n qubits, with $n-1$ *control* qubits, and the remaining qubit being the *target*. If all control qubits are set to $|1\rangle$, then the target qubit is flipped; otherwise, it is left undisturbed. For $n = 2$, MCT reduces to the fundamental Controlled-NOT Gate (**CX**).

We next present a brief overview of the basic quantum algorithms leveraged in QIA.

2.2 Quantum Approximate Optimization Algorithm (QAOA)

QAOA is a hybrid algorithm that combines classical and quantum components and is tailored to find approximate solutions to optimization problems [33]. QAOA operates through a sequence of quantum and classical steps. The quantum part involves initializing qubits in the uniform superposition state $|+\rangle$, and then applying a specialized quantum circuit (configured with $2p$ parameters) on the initial state p times. A quantum computer is used to evaluate the objective function, while a classical optimizer is used to update the $2p$ parameters. This iterative process is repeated until the classical optimizer converges. The protocol, as outlined in [16], is shown in Figure 4.

As might be expected, the approximation quality of QAOA improves with p, but the circuit depth also grows linearly with p. Therefore, p is usually set to a small value to balance solution quality and quantum feasibility. In fact, even at the lowest circuit depth (p = 1), QAOA has non-trivial provable performance guarantees [34], and hence p = 1 has been used as a common assignment in the literature [53]. However, the literature also suggests that a logarithmic depth is anticipated to surpass classical optimizers [56]. Therefore, in our evaluation, we vary p in the range 1 to $\lceil \log(|I|) \rceil$, where I is the list of candidate indexes.

2.3 Grover Search (GS)

GS is a quantum algorithm designed to solve the unstructured search problem with high probability (WHP) [37]. Specifically, given an unordered list of N items, GS identifies a desired item WHP, using $\mathcal{O}(\sqrt{N})$ iterations, as compared to the $\mathcal{O}(N)$ probes incurred by the classical algorithms. During each iteration, GS leverages quantum properties to simultaneously check all N items, resulting in a quadratic speed-up compared to classical methods.

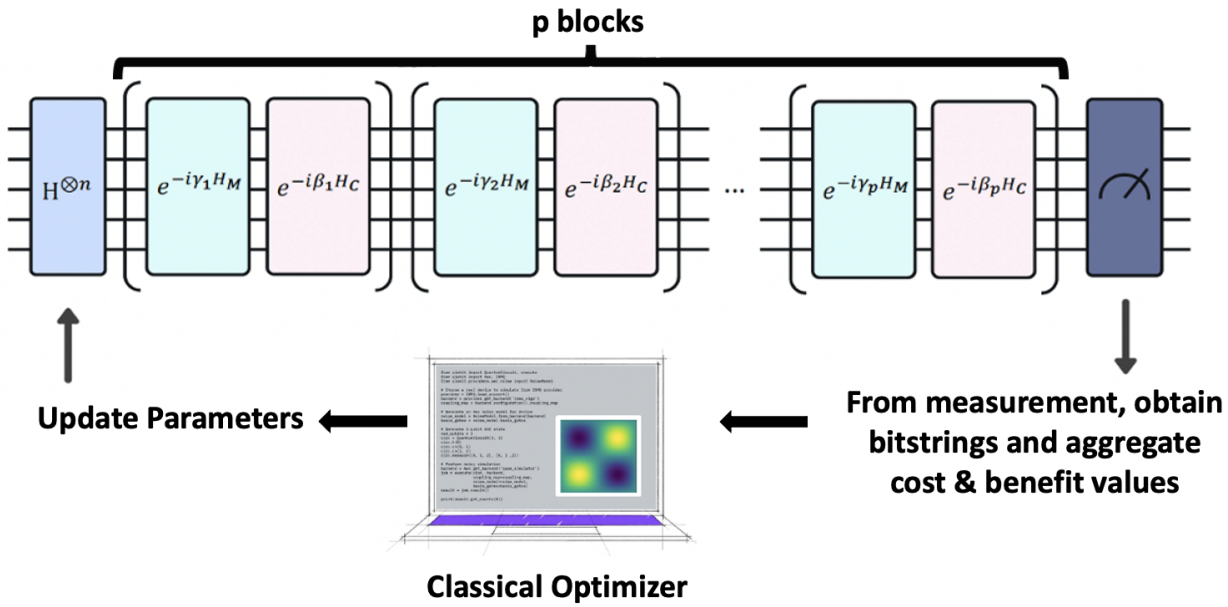


Figure 4: The QAOA Protocol [16]

The GS algorithm involves three key operators: *state preparation*, *quantum oracle*, and *diffusion*. The state preparation operator creates a quantum superposition state encompassing an exponential solution space. The quantum oracle enhances each candidate solution with problem-specific data, evaluates the cost function, and identifies qualifying candidates. The diffusion operator amplifies the measurement probability of the qualified candidates. Notably, the state preparation is a one-time activity. The combined application of the quantum oracle followed by the diffusion operator is termed a **Grover Iteration**. It is invoked multiple times to identify a qualifying candidate with a probability exceeding 0.5.

The key challenges in designing an index selection scheme based on GS include: (1) Creating an efficient problem-specific quantum oracle; (2) Identifying the precise number of Grover Iterations required for the GS algorithm to work appropriately; and (3) Boosting the success probability from 0.5 to the user-desired δ. In Section 5, we present a novel approach to constructing the quantum oracle utilizing various quantum concepts implementable through standard quantum gates. Additionally, we address the remaining challenges by adapting the Generalized Grover Search (GGS) [20, 23] algorithm and the Powering Lemma [40], which enable us to optimize the algorithm's performance. These modules are discussed in detail in Section 5.2.

2.4 Shots

An operational parameter that influences the solution quality of quantum algorithms is *"shots"(S)*. Shots indicate the number of times a quantum algorithm is executed, with increased shots providing more accurate and reliable results at the expense of consuming more quantum resources. In our experiments, we empirically identify the ideal number of shots for the proposed QIA schemes.

3 Problem Framework

The Index Advisor problem that we consider here is the following: Given an SQL query workload $\mathcal{Q}$ on a relational database instance $\mathcal{D}$, recommend a configuration of indexes that maximizes the performance benefit for the workload while adhering to the following constraints: (1) **Space Constraint:**

The configuration must fit in a user-specified storage budget; and (2) **Validity Constraint:** The configuration must satisfy validity requirements, which could be (a) intrinsic – for instance, at most one *clustered* index per relation, or (b) extrinsic – for instance, mandatorily add all indexes listed in a pre-specified base configuration.

In the above problem definition, the performance benefit of a configuration is measured as the reduction in the estimated execution time of the workload compared to the base configuration. Further, since contemporary database systems (e.g. DB2) typically build, by default, a clustering index on the primary key column of each relation, we assume that the base configuration comprises these indexes. Therefore, our objective is restricted to selecting the additional *unclustered* indexes.

Index Advisor Pipeline

The index advisor pipeline encompasses a sequence of tasks to find a beneficial configuration of constraint-compliant indexes. The tasks, shown pictorially in the top pipeline of Figure 2 as Steps 1 through 4, are the following:

1. **Candidate Generation:** This task entails identifying candidate indexes to improve the SQL workload performance. Techniques such as analyzing query predicates and recognizing common query patterns, are employed to create a comprehensive pool of potentially beneficial indexes.

2. **Cost Evaluation:** This step computes the individual storage and maintenance (due to updates) overheads of the candidate indexes. Storage overheads serve to model the index cost, whereas maintenance overheads are factored into the benefit calculations of the next stage.

3. **Benefit Computation:** The overall improvement of the query workload execution time due to the presence of each index is computed, typically via the query optimizer module. Specifically, the cumulative improvement in query response times is weighed against the increase in index maintenance overheads.

4. **Index Selection:** This final step aims to find the configuration among the candidate indexes that maximizes the benefit while respecting the storage and validity constraints.

Our study employs classical strategies for the first three tasks in the pipeline and uses the quantum platform only for the final computationally-intensive index selection task. Specifically, we use DB2 Index Advisor [55] as the exemplar classical technique.

In this formulation, given a set of indexes $I = \{i_0, i_1, \cdots, i_{n-1}\}$, each with its storage overhead $W = \{w_0, w_1, \cdots, w_{n-1}\}$ and time benefit $V = \{v_0, v_1, \cdots, v_{n-1}\}$, index selection in Step 4 is modeled as the following constrained optimization problem:

$$\max \sum_{i=0}^{n-1} x_i v_i \quad \text{s.t.} \quad \sum_{i=0}^{n-1} x_i w_i \leq W_{max} \tag{2}$$

where the solution to the problem is represented by an array, denoted as X, consisting of elements $\{x_0, x_1, \cdots, x_{n-1}\}$, each taking binary values of 0 (exclusion) or 1 (inclusion). The objective is to select and recommend a configuration of indexes that maximizes the performance benefit for the workload while adhering to the storage/validity constraints.

Table 1: Parameters

Type	Symbol	Description	Domain
Input	I	List of Indexes	$[i_0, \cdots, i_{n-1}]$
Input	W	List of Storage Costs	$[w_0, \cdots, w_{n-1}]$
Input	V	List of Time Benefits	$[v_0, \cdots, v_{n-1}]$
Input	W_{max}	Storage Budget	$(0, \sum_{i=0}^{n-1} w_i]$
Input	V_{max}	Maximum Benefit	$[0, \sum_{i=0}^{n-1} v_i]$
Input	L	# of Possible Configurations	$2^{\lvert I \rvert}$
Input	S	# of Shots	$\mathbb{Z}^+$ [Pos. Integer]
OQIA	p	Repetition Depth of QAOA	$\{1, \cdots, \log(\lvert I \rvert)\}$
SQIA	δ	Desired Optimality Probability	$[0.5, 1)$
SQIA	ϵ	Failure Probability	$1 - \delta$
SQIA	λ	Step size for GGS	$[1, 1.33]$
SQIA	α	Timeout Control of GGS	$\mathbb{Q}^+$ [Pos. Rational]
SQIA	Max_{iter}	Upper bound on Grover Iteration	$\lfloor \alpha \cdot \sqrt{L} \rfloor$
SQIA	R	# of Repetitions of GGS	$\lceil \log(1/\epsilon) \rceil$
SQIA	C_x	Candidate Configuration wrt. x	$x = \{0, 1\}^n$
SQIA	$W(C_x)$	Aggregate cost of C_x	$\mathbb{Z}^+$ [Pos. Integer]

3.1 Notation

The various input and algorithmic parameters used in the sequel, together with their notation, are summarized in Table 1.

In the subsequent sections, we describe the proposed Optimization-based (OQIA) and Search-based (SQIA) quantum schemes to implement Stage 4 of the Index Advisor pipeline.

4 Optimization-based QIA (OQIA)

In the first step of OQIA, the index selection optimization problem, as defined in Equation 2, is transformed into a Quadratic Unconstrained Binary Optimization (QUBO) instance. We then map the QUBO instance to an Ising Hamiltonian using the transformations outlined in [44]. This two-step process helps convert the index selection problem instance into a suitable format for consumption by Quantum Approximate Optimization Algorithm (QAOA).

QUBO for Index Selection

QUBO problems feature binary decision variables, quadratic objective functions, and no constraints – the objective is to identify the assignment of binary variables that minimizes the quadratic objective function. For the index selection problem, we essentially use the QUBO reformulation technique presented in [32] with some minor modifications. The process operates as follows: First, Equation 2 is converted from a maximization task into a minimization task – this is trivially achieved by changing the sign of the objective function:

$$\sum_{i=0}^{n-1} x_i v_i \rightarrow -\sum_{i=0}^{n-1} x_i v_i \tag{3}$$

Next, the storage space constraint is internalized to make the optimization objective constraint-free. This process requires additional machinery, specifically the introduction of an auxiliary term, denoted

$\mathcal{B}$ in the optimization objective. This $\mathcal{B}$ term undergoes dynamic adjustments in each iteration by the optimizer in response to the solution's characteristics.

For starters, the original solution array X of length n bits is expanded to include an additional m bits, resulting in an expanded array denoted $X^B = [x_0, \cdots, x_{n-1},\ b_n, \cdots, b_{n+m-1}]$. The additional bits b_j are formed from the binary representation of $\mathcal{B}$, that is, $\mathcal{B} = \sum_{j=n}^{n+m-1} 2^j b_j$.

Next, the objective function is updated to enable the optimizer to simultaneously optimize the values of X and $\mathcal{B}$. To do so, the $\mathcal{B}$ is subtracted from the storage constraint, the resultant is squared, and then multiplied with a large positive number A. The full expression is:

$$A \cdot \left(W_{max} - \sum_{i=0}^{n-1} x_i w_i - \mathcal{B} \right)^2 \tag{4}$$

The updated cost function is obtained by adding Equations 3 and 4:

$$\begin{aligned} C(X^B) &= A \cdot \left(W_{max} - \sum_{i=0}^{n-1} x_i w_i - \mathcal{B} \right)^2 - \sum_{i=0}^{n-1} x_i v_i \\ &= A \cdot \left(W_{max} - \sum_{i=0}^{n-1} x_i w_i - \sum_{j=n}^{n+m-1} 2^j b_j \right)^2 - \sum_{i=0}^{n-1} x_i v_i \end{aligned} \tag{5}$$

resulting in the following QUBO objective function:

$$\min_{X^B} \left(A \cdot \left(W_{max} - \sum_{i=0}^{n-1} x_i w_i - \sum_{j=n}^{n+m-1} 2^j b_j \right)^2 - \sum_{i=0}^{n-1} x_i v_i \right) \tag{6}$$

The minimization of the above function requires the first term to go to 0, implying that $\sum_i x_i w_i \leq W_{max}$, which is the storage constraint. Given a zero-valued first term, the function minimization is determined by the second term, which is the aggregate benefit. Therefore, benefit maximization is achieved subject to meeting the storage budget. Next, we map the generated QUBO instance (Equation 6) to an Ising Hamiltonian using the scheme outlined in [44].

4.1 QUBO to Ising Hamiltonian

Quantum computers require problems to be formulated as Hamiltonians, which describe a physical system's energy in terms of operators or matrices. Therefore, we need to design a Hamiltonian that encapsulates the adjusted cost function (Equation 5). Post this step, we can identify the system's lowest energy state, commonly referred to as the ground state, using the QAOA optimization algorithm – this state captures the best index configuration.

We map the generated QUBO instance (Equation 6) to an Ising Hamiltonian using the scheme outlined in [44]. Particularly, we first convert the binary variables $x_i \in \{0, 1\}$ and $b_j \in \{0, 1\}$ to spin variables $z_i \in \{-1, +1\}$, and then promote the spin variable z_i to Pauli Z_i operators. This is achieved with the following mapping [44]:

$$x_i \rightarrow \frac{I - Z_i}{2} \tag{7}$$

where (a) I is the identity matrix of order $n + m$, and (b) Z_i is the matrix that results from the tensor product of $n + m$ matrices with the following structure: At position i they feature the Pauli Z matrix, and at all remaining positions, contain the identity matrix I.

An essential feature of the above mapping is that it is "eigen-preserving" in that the eigenstates corresponding to the minimum and maximum eigenvalues stay at the same "places". Therefore, finding the ground state of the Ising Hamiltonian corresponds to minimizing the corresponding QUBO problem. In the last step, the resulting Hamiltonian is fed to the standard QAOA algorithm implementation [8] for optimization.

4.2 OQIA Resource Scaling

The number of qubits required to implement the OQIA pipeline is determined by the number of binary variables in the QUBO objective function. As shown in Equation 6, it comprises of two sets of binary variables – (a) The n binary variables $\{x_0 \cdots x_{n-1}\}$ representing indexes, and (b) The m binary variables $\{b_n, \cdots, b_{n+m-1}\}$ composing $\mathcal{B}$ which is inherently upper-bounded by W_{max}. Assuming that the storage budget W_{max} can fit within a 32-bit integer, the qubit count scales *linearly* with the number of indexes.

5 Search-based QIA (SQIA)

We now turn our attention to solving the index selection problem as an enumerative search over the exponential configuration space using the Grover Search (GS) algorithm. We first leverage the quantum *superposition* property to generate the exponential space of candidate index configurations, incurring only logarithmic qubit overhead. Next, we present a novel algorithm for constructing a quantum oracle for the index selection problem. Our oracle harnesses the power of quantum *entanglement* and efficiently loads the problem instance into the qubit *phases* – this is a shift from the normal practice of loading into qubit basis states. Then, we provide procedures for configuring the GS algorithm to overcome various practical challenges. Finally, we analyze the computational complexity of our approach and demonstrate that SQIA potentially exhibits a *linear* qubit scalability with index set size while preserving the quadratic speed-up offered by the GS algorithm.

5.0.1 Generate Candidate Configurations

Given the set of candidate indexes I, the total number of possible candidate configurations is $L = 2^{|I|}$. In classical computing, we need exponential time or exponential resources to access all the configurations. However, in quantum computing, we could use the *superposition* property to simultaneously load these L candidate configurations in I qubits using just I quantum **Hadamard (H)** gates.

To achieve this objective, we initialize a quantum circuit with $n = |I|$ qubits, where each qubit maps to an element of the index set I. Initially, all the qubits are in the $|0\rangle$ quantum state. Now, in order to implicitly generate the powerset of I, we apply the quantum **H** gate on all qubits. As mentioned in Section 2, the **H** gate takes a qubit in the $|0\rangle$ state to an equal superposition state i.e., $\frac{|0\rangle+|1\rangle}{2}$. Now, when the **H** gate is applied individually to n qubits, we get:

$$\begin{aligned} H|0\rangle_0 \otimes \cdots \otimes H|0\rangle_{|I|-1} &= \frac{|0\rangle + |1\rangle}{2} \otimes \cdots \otimes \frac{|0\rangle + |1\rangle}{2} \\ &= \frac{1}{L} \sum_{x=0}^{L-1} |x\rangle \end{aligned} \tag{8}$$

In the above equation, $|x\rangle$ corresponds to the binary representation of the corresponding integer, and therefore Equation 8 represents an equal superposition of all L candidate index configurations. To

visualize, consider the binary representation of any integer $x \in \{0, L-1\}$. For every bit $j \in \{1, |I|\}$, if $x_j = 1$, then include the index i_j in the corresponding candidate configuration, otherwise not.

5.1 Quantum Oracle for Index Selection

Having created all candidate configurations, we now move on to showing how the quantum *entanglement* property can be leveraged for assigning storage costs and time benefits to the candidate configurations, and maintaining compliance with the storage constraint. This is done in conjunction with qubit phase manipulation. We present novel building blocks that construct a quantum oracle for the index selection problem, and this oracle is subsequently used in the GS algorithm to identify the qualifying configurations.

5.1.1 Encoding Storage Costs (Index Weights)

Given a candidate configuration C_x, $W(C_x)$ represents the aggregate cost of its constituent indexes. For instance, if $C_x = \{i_0, i_2\}$, then its cost $W(C_x) = w_0 + w_2$. Further, the maximum possible cost is the cost of the configuration that includes all the candidate indexes. Therefore, we need $m = \lceil \log_2(\sum_{i=0}^{|I|-1} w_i) \rceil$ qubits to cover all the costs that could appear during the execution.

Direct Approach

The simplest way to associate costs to candidate configurations is to first pre-compute the costs of all candidate configurations. Then, to single out each configuration (present in the uniform superposition) using the quantum **NOT (X)** gate, and insert the corresponding cost in the dedicated cost qubits. But this requires: 1) Pre-computing costs for an exponential number of configurations; and 2) Applying an exponential number of quantum **X** gates to identify each candidate configuration uniquely.

Angle Encoding

Angle encoding itself doesn't differentiate between states – any data point maps to a valid angle on the Bloch sphere. It offers no inherent mechanism to "mark" the target state within the superposition, which is crucial for the Grover diffusion operator to amplify its probability.

Amplitude Encoding

Amplitude encoding involves using the amplitudes of quantum states to represent data, where the amplitudes are assigned to basis states according to the data values. However, it's unclear how to create a quantum oracle to identify and mark target states using this encoding. Even if such an oracle were somehow created, after applying Grover's diffusion operator once, all state amplitudes would change, effectively altering the data. Further, this mutation would vary depending on the number of qualifying states. Consequently, since the original data is mutated with an unknown amount, it is not clear how the GS algorithm will proceed to identify the qualifying states.

Qsample Encoding

Qsample encoding represents a classical probability distribution using a real-valued amplitude vector. Each element in the vector corresponds to a classical data point, and its amplitude signifies the square root of the associated probability. While this vector could be used to initialize a quantum state, it lacks

the capability to "mark" the target state during the Grover iteration, which is essential for amplifying its probability.

Phase-based Approach

Given the above problems with the popular data encoding approaches, we design an alternate strategy based on qubit *phase* manipulation. The state of a qubit, as depicted by Equation 1, has three components: the basis states ($|0\rangle$ and $|1\rangle$), the amplitudes (α and β) associated with the basis states , and the relative phase (γ) of the $|1\rangle$ state. We convert the classical costs into suitable angles γ (in Fourier basis), and then load them as the relative phase of the qubits using the quantum **Controlled-Phase (CP)** gate. This strategy helps us to intrinsically compute and associate the costs of an exponential number of candidate configurations using just $m \cdot |I|$ two-qubit **CP** gates.

Algorithm 1 load_cost_SQIA

Require: $W = [w_0, w_1, \cdots, w_{n-1}]$ ▷ Input costs
1: $n \leftarrow |W|$ ▷ # of Indexes
2: $m = \lceil \log_2(\sum_{i=0}^{n-1} w_i) \rceil + 1$
3: $qc \leftarrow QuantumCircuit(n+m)$
4: $qc.h(m)$ ▷ Generate Equal Superposition
5: **for** $i \in range(n)$ **do**
6: $\quad \theta_i = \frac{w_i}{2^m} \cdot 2\pi$
7: $\quad$ **for** $j \in range(m)$ **do**
8: $\quad\quad \gamma_j = 2^{(m-j-1)} \cdot \theta_i$
9: $\quad\quad qc.cp(\gamma_j, i, j)$
10: **return** qc

The above process is summarized in Algorithm 1, which takes a list of costs W as input. As mentioned earlier, we need m qubits to encode the costs of all candidate configurations. However, an additional qubit is allocated (Line 2) for storing the *sign* of the costs. Specifically, a state $|0\rangle$ in the sign qubit represents a positive cost. The significance and necessity of this supplementary sign qubit will become evident in Section 5.1.3. Subsequently, we instantiate a quantum circuit qc with the capacity to accommodate both index (n) and cost ($m+1$) qubits – all these qubits are initialized to the equal superposition state $|+\rangle$ using the **H** gate.

Now consider an index $i \in I$ with cost w_i. To load w_i in the m cost qubits (excluding the sign qubit, since by default it is in the positive state), we compute the phase angles $\gamma_1, \cdots, \gamma_m$ for each qubit. For ease of understanding, we decompose this angle computation into two parts:

Rotation Angle: The core angle of rotation θ_i is the rotation relative to the full rotation 2π. It depends on w_i and the number of qubits, namely m, on which this cost is encoded.

Rotation Frequency: The frequency of rotation depends on the qubit position – 1^{st} qubit is rotated by an angle $2^{m-1} \cdot \theta_i$, and this angle is progressively halved for the following qubits, i.e., the j^{th} qubit is rotated by an angle $\gamma_j = 2^{m-j-1} \cdot \theta_i$

Next in Lines 7 – 9 of Algorithm 1, we apply a controlled Z-axis rotation by an angle γ_j using the **CP** gate. Here, the index qubit i is the control qubit, and the j^{th} cost qubit is the target. The application of the **CP** gate *entangles* the index qubits with the cost qubits. And the angle of rotation γ_i loads the costs in the phase of these qubits. We repeat this process for all indexes in I.

Phase Loading Example. To make clear the above process of loading costs via phases, consider the candidate configuration $C_x = \{i_0, i_2\}$ for the problem instance shown in Figure 1. Here, the aggregate cost $W(C_x) = w_0 + w_2$, and the index qubit assumes the quantum state $|0000101\rangle$ (in superposition). It therefore triggers the conditional rotations corresponding to index qubits i_0 and i_2. Index qubit i_0 adds a relative phase γ_j to the j^{th} cost qubit (m_j), computed as:

$$\gamma_j = 2^{(m-j-1)} \cdot 2\pi \cdot \frac{w_0}{2^m} \tag{9}$$

producing a quantum state $|m_j\rangle = \frac{|0\rangle + e^{i \cdot 2^{-(j+1)} \cdot 2\pi \cdot w_0} |1\rangle}{2}$. Similarly, index qubit i_2 adds a phase angle $\gamma_j = 2^{-(j+1)} \cdot 2\pi \cdot w_2$ to m_j. Therefore, the final state of m_j is:

$$|m_j\rangle = \frac{|0\rangle + e^{i \cdot 2^{-(j+1)} \cdot 2\pi \cdot (w_0 + w_2)} |1\rangle}{2} \tag{10}$$

With the above rotations, the desired aggregate cost is loaded in the phase of the cost qubits.

5.1.2 Encoding Benefits

The encoding of benefits mirrors the encoding of costs, and Algorithm 1 is reused with minor modifications: The list of benefits V is passed instead of the costs W, and the number of qubits needed are $v = \lceil \log_2(\sum_{i=0}^{n-1} v_i) \rceil + 1$. The rest of the algorithm is followed as-is.

5.1.3 Encoding Storage Constraint

To encode the storage space constraint, we subtract W_{max} from the cost qubits for all L configurations in superposition. For this, we encode the negative of the storage budget ($-1 \cdot W_{max}$) as angles ($\gamma_1, \cdots, \gamma_m$), and apply a single qubit rotation gate (quantum **Phase (P)** gate) on each cost qubit. The **P** gate is used instead of **CP** to make the rotation independent of the index qubits, thereby applying it uniformly across all the superposed index states. After this step, all configurations that satisfy the storage space constraint will have a negative cost loaded as a relative phase in their cost qubits.

5.1.4 Extracting Signed Costs

As discussed in Section 2, relative phases are not directly measurable. To make them measurable, we apply the *inverse Quantum Fourier Transform* (QFT) algorithm [47] to the cost qubits and *transform the costs from the phase to the basis state of the qubits*. A key point to note here is that the costs are encoded in the angles of a periodic function ($e^{i\gamma}$) with a period 2π, and therefore, when the inverse QFT operation is performed, the costs are obtained in *two's complement* format. Accordingly, costs associated with the configurations satisfying the storage constraint are all either 0 or negative, and the negative configurations can be easily identified since their sign qubit will be in quantum state $|1\rangle$. Further, the 0 cost configurations are identified by ignoring the sign qubit and checking if the rest of the cost qubits are in $|0\rangle$ state. These checks are easily encoded in the quantum circuit using **MCT** gates to detect and signal the qualifying configurations.

5.1.5 Encoding Benefit Constraint

Our objective is to identify the index configuration having the maximum benefit from among the candidate configurations satisfying the storage constraint. However, since we do not know the optimal

benefit value, a candidate target benefit value (V_{target}) is chosen (as described later in Section 5.2). We next encode V_{target} as we encoded the storage constraint – take the negative of the target benefit ($-1 \cdot V_{target}$) as angles ($\gamma_1, \cdots, \gamma_m$), and apply a single qubit rotation gate (quantum **Phase (P)** gate) on each benefit qubit. This process subtracts V_{target} from the accumulated benefit of all the configurations in a quantum superposition state.

To extract signed benefit values associated with each configuration, we will apply the same procedure as was done earlier for the storage constraint in Section 5.1.4. Here, all the qualifying candidate configurations will have a positive benefit encoded in the benefit qubits (after substrating V_{target}) and can be easily identified using quantum **X** gates – to check if the sign qubit is in the quantum $|0\rangle$ state and signal the qualifying configurations.

5.1.6 Identify Candidate Configurations

Till now, we have separately identified the configurations that satisfy the storage or target benefit constraints. But the configurations that qualify both these constraints simultaneously are the real candidate configurations. To achieve this, we need 3 additional qubits in the quantum circuit – one qubit to record the storage constraint satisfaction (W_{sig}) and another to record the target benefit constraint satisfaction (V_{sig}). Then, we will conjugate the two constraint satisfaction qubits and signal the final candidate qualification in the third qubit (out).

The cost and benefit constraint satisfaction is signaled, as discussed in the previous sections 5.1.4 and 5.1.5, respectively. Now, the conjugation is easily achieved by using a three-qubit **Control-Control-Not (CCX)** gate with W_{sig} and V_{sig} as the control qubits and C_{sig} as the target qubit. The **CCX** gate will flip the C_{sig} qubit only when both the W_{sig} and V_{sig} qubits are in the quantum state $|1\rangle$.

5.1.7 Composing the Quantum Oracle

For a given instance of the index selection problem, we compose the above modules (cost encoding, benefit encoding, constraint encoding, cost extraction), combine their circuits, and construct the quantum oracle. The oracle integrates with the GS algorithm and signals the qualifying configurations. The construction details are summarized below in Algorithm 2.

Algorithm 2 config_sel_oracle

Require: List: (I, W, V) and Int: (W_{max}, V_{target})

1: Encode Storage Costs (Section 5.1.1)
2: Encode Benefits (Section 5.1.2)
3: Encode Storage Constraint (Section 5.1.3)
4: Extract Signed Costs (Section 5.1.4)
5: Encode Benefit Constraint (Section 5.1.5)
6: Signal Qualifying Configurations (Section 5.1.6)
7: **return** qc

5.1.8 Oracle Construction Example

To include one more example for oracle construction, taking a new index selection problem instance, having the same number of indexes as used in the main paper, but with different cost and benefit values. Consider the following index selection problem instance (we will refer to this as **TR_I2**): $I = [i_0, i_1]$, $W = [1, 4]$, $V = [2, 4]$, and $W_{max} = 4$. The quantum oracle for this problem is shown in Figure 5, and

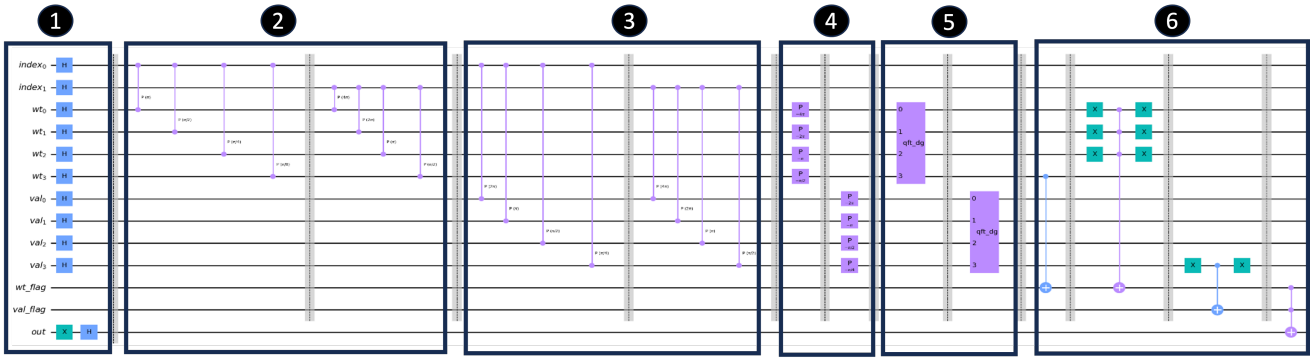


Figure 5: Quantum Circuit of SQIA Oracle for Problem Instance **TR_I2**

the construction process is summarized below:

Step 0: Count the number of qubits needed to store the indices $n = |I| = 2$, the cost $m = \lceil \log_2(\sum_{i=0}^{n-1} w_i) \rceil + 1 = 4$ and the benefit $v = \lceil \log_2(\sum_{i=0}^{n-1} v_i) \rceil + 1 = 4$. Additionally, one qubit is required for each of the following: encoding the storage constraint, encoding the target benefit constraint, and flagging the qualifying configurations. Note, the target benefit constraint is an additional constraint added in SQIA and is explained later in Section 5.2. Therefore, the total number of qubits needed is 13.

Step 1: Initialize the index, cost, and benefit qubits in an equal superposition $|+\rangle$ state using the quantum **H** gate and the output qubit in the $|-\rangle$ quantum state. Although the index, cost, and benefit qubits are initialized using the same quantum gate, their interpretations vary depending on how these qubits will be used in the rest of the quantum circuit. Specifically, the index qubits are viewed jointly; hence, as shown earlier, they generate all the 4 possible index configurations after initialization. The remaining qubits are viewed independently – the cost and benefit qubits are kept in state $|+\rangle$ to load and process the data in the relative phase of the qubit while the output qubit is in state $|-\rangle$ to flag the qualifying index configurations through phase kickback [47] to the index qubits.

Step 2: For each $w_i \in W$, we calculate the phase angles and load them into the relative phase of the cost qubits by applying the Controlled Phase **CP** gate. For example, for $w_1 = 1$, the rotation angle $\theta_1 = \frac{2*\pi}{2^m} \cdot w_1 = \frac{\pi}{8}$ and therefore the phase angles are $\gamma_1 = \pi, \gamma_2 = \frac{\pi}{2}, \ \gamma_3 = \frac{\pi}{4} \ and \ \gamma_4 = \frac{\pi}{8}$.

Step 3: Similarly, for each benefit $v_i \in V$, we calculate the phase angles and load them into the relative phase of the benefit qubits.

Step 4: Next, we apply the storage and the target benefit constraints. For example, for $W_{max} = 4$, we compute the phase angle for -2 and apply it to all cost qubits using a quantum **Phase (P)** gate.

Step 5: Now, to retrieve the signed cost and benefit values, the inverse QFT operation is performed on the cost and benefit qubits.

Step 6: Finally, the qualifying index configurations are signalled via the output qubit.

5.1.9 Correctness of SQIA Oracle

To demonstrate the correctness of the SQIA quantum Oracle, we conduct a step-by-step validation of the individual logical operations performed throughout the construction process. Specifically, we observe the quantum state of the Oracle after every logical operation is applied in the quantum circuit and validate it. To achieve this, we execute the intermediate quantum circuit constructed after every logical operation and project the observed state of the qubits as a histogram (obtained by measuring the quantum circuit 5000 times). For ease of presentation, we will use the index selection problem instance **TR_I2** whose Oracle construction process is summarized in Section 5.1.8. Further, to enable

validation, we will make the following changes in the generated Oracle (shown in Figure 5):

1. Add quantum measurement gates to the index, cost, benefit and the output qubits to measure their quantum states.

2. Change the initialization of the output qubit from $|-\rangle$ quantum state to $|0\rangle$. This will help us capture the configuration qualification status in the output qubit instead of passing it to the index qubits via phase kickback [6].

Quantum State After Data Load: After Steps 0, 1, 2 and 3 of oracle construction (discussed in Section 5.1.8), the input index selection problem instance should be loaded into the quantum circuit. Specifically, the quantum state of the qubits at this point should have all the exponential number of index configurations along with their aggregate cost and benefit appearing simultaneously with equal probability in a quantum superposition state. To validate this, we cut the Oracle quantum circuit (shown in Figure 5) at this point and apply inverse Quantum Fourier Transform operation on the cost and benefit qubits (to make their quantum state measurable) and then apply the measurement operations on all the qubits. The truncated quantum circuit with measurement operation is shown in Figure 6.

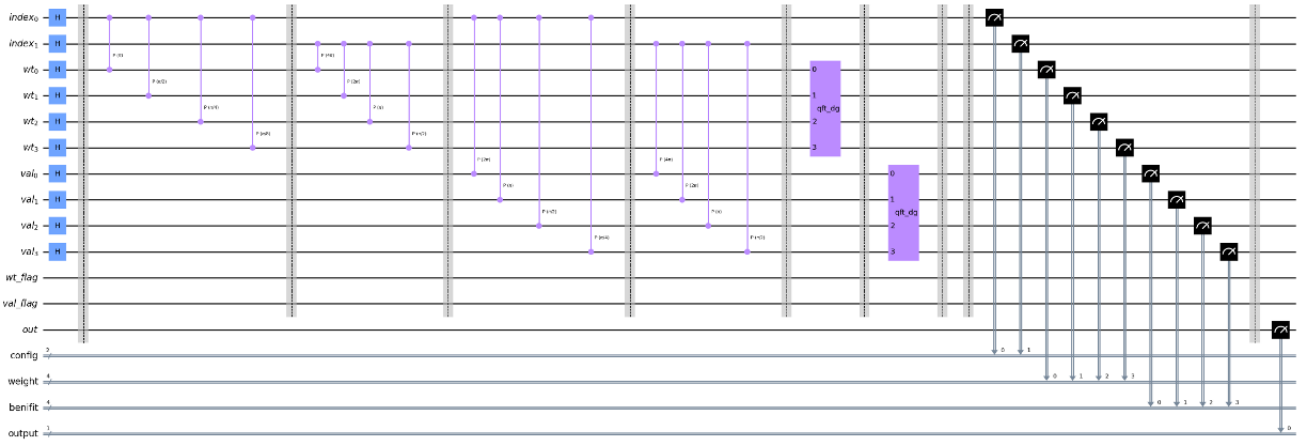
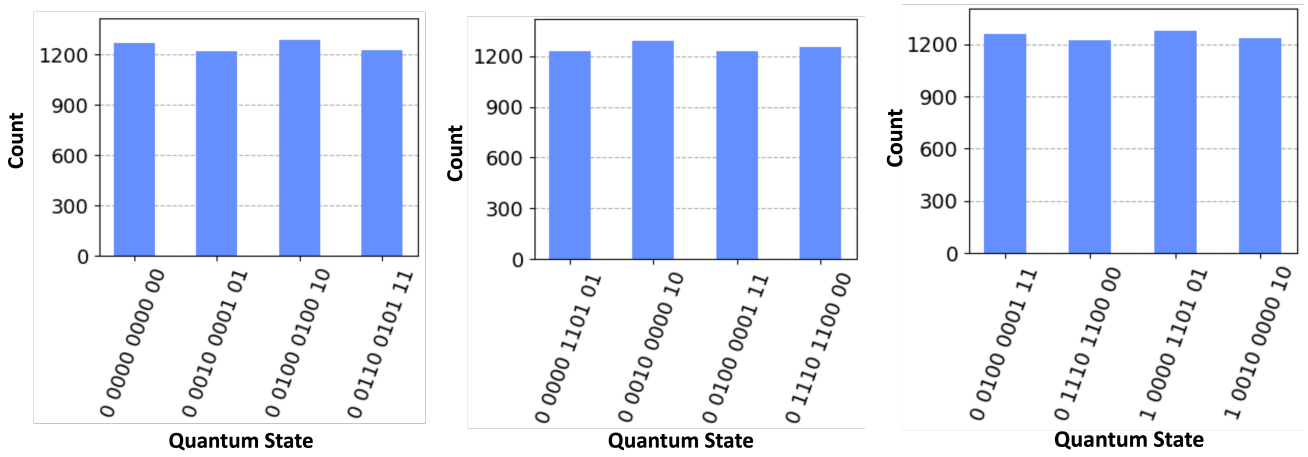


Figure 6: SQIA Oracle with measurement after Data Load operation

We then execute this quantum circuit and observe the state of the qubits. After one execution, we get the information of only one randomly selected quantum state from among all the states that exist simultaneously in the quantum superposition state. Therefore, to obtain the full picture, we repeat the execution 5000 times and plot a histogram over the observed quantum states, which is shown in Figure 7a. The x-axis of the histogram represents the observed quantum states, while the y-axis shows the number of times each quantum state was observed. Note, an observed quantum state on the x-axis is grouped into four chunks of bits and its interpretation is summarized in Figure 8. Further, the aggregate cost and benefit values are projected in their *two's complement* notation. For an easy understanding, the observed quantum states after data load are interpreted and summarized in Table 2. We can see that, as expected, all the four possible index configurations appear in the quantum superposition state with an equal probability (since each quantum state was randomly sampled almost equal number of times), validating the correctness of the data load operation via qubit *phase* manipulation.

Quantum State After Applying Constraints: Next we apply the storage and benefit constraints (Step 4 of Figure 5) and again execute and measure the quantum circuit. The histogram over the



(a) Quantum State After Data Load

(b) Quantum State After Applying Constraints

(c) Quantum State After Checking Qualification

Figure 7: Quantum State of SQIA Oracle after Different Operations

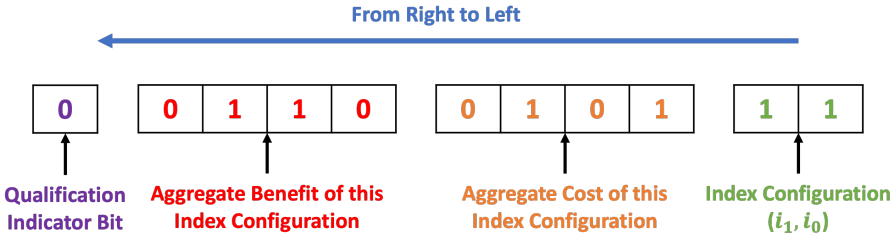


Figure 8: Observed Quantum State

Index Configurations (i_1 i_0)	After Data Load			After Constraints			After Qualification Test		
	Cost	Benefit	Qualify	Cost	Benefit	Qualify	Cost	Benefit	Qualify
0 0	0	0	0	-4	-2	0	-4	-2	0
0 1	1	2	0	-3	0	0	-3	0	1
1 0	4	4	0	0	2	0	0	2	1
1 1	5	6	0	1	4	0	1	4	0

Table 2: Evolution of Quantum States

observed quantum states is presented in Figure 7b and is interpreted again in Table 2. We can see that as expected, the constraints are applied to all the index configurations simultaneously with just one application of the quantum **P** gate. Hence validating the correctness of this part of the SQIA oracle.

Quantum State After Checking Qualification: Finally, we apply the constraints satisfaction checks (Step 6 of Figure 5) to record the qualification of the index configuration in the output qubit. We then again execute and measure the quantum circuit. The histogram over the observed quantum states is shown in Figure 7c and is interpreted again in Table 2. We can see that, only two index configurations qualify and are indicated by setting their corresponding output qubit. Specifically, index configurations 1) comprising of only index $\{i_0\}$ and 2) comprising of only index $\{i_1\}$ qualify, while the remaining two configurations are marked as invalid, since the empty index configuration failed the target benefit validation check, while the index configuration comprising of both the indexes $\{i_1, i_0\}$ failed the storage

constraint check. *Hence validating the correctness of the SQIA oracle.*

5.2 Finding Optimal Index Configuration

We now turn our attention to the selection process, implemented via the **SQIA _search** procedure outlined in Algorithm 3. It takes an index selection problem instance comprising of $[I, W, V, W_{max}]$ as input and produces an optimal index configuration with a user-settable success probability $\delta \in [0.5, 1)$.

Since the precise benefit accrued by the optimal configuration is initially unknown, Algorithm 3 employs a recursive halving strategy, starting with V_{max}, the upper bound, to identify this value. In each iteration, the $find\_opt\_config$ procedure (shown in Algorithm 4), is used to identify the optimal index configuration achieving a benefit greater than or equal to the targeted benefit value. A failure indicates that the target benefit is too large to be feasible. The first successful identification indicates a transition from infeasible to feasible range, and we now again carry out a recursive halving within this transition range to finally identify the optimal configuration.

At its core, the $find\_opt\_config$ method relies on the GS algorithm. However, as mentioned in Section 2, for the effective utilization of the GS algorithm, three key elements must be provided: 1) an appropriate quantum oracle, 2) the precise number of Grover Iterations, and 3) an enhancement of its success probability from 0.5 to the desired δ. While Section 5.1 guides the construction of the corresponding quantum oracle for any target benefit value, the following discussion will address the remaining two issues.

Algorithm 3 SQIA _search

Require: List: (I, W, V), Int: W_{max}
1: $V_{target} = V_{max}$
2: $V_{min} = 0$
3: $opt_{ind} = null$
4: **while** $V_{min} < V_{target}$ **do**
5: $\quad P_{res} \leftarrow find\_opt\_config(I, W, V, W_{max}, V_{target}, \delta)$
6: $\quad$ **if** $P_{res}[qualify] == True$ **then**
7: $\quad\quad opt_{ind} = P_{res}[ind]$
8: $\quad\quad V_{min} = V_{target}$
9: $\quad\quad V_{target} = \lceil (P_{res}[val] + V_{max})/2 \rceil$
10: $\quad$ **else**
11: $\quad\quad V_{target} = \lceil (V_{min} + V_{target})/2 \rceil$
12: **return** opt_{ind}

5.2.1 Finding precise number of Grover Iterations

To calculate the number of Grover Iterations (Oracle + Diffusion), the GS algorithm needs precise knowledge of the number of qualifying index configurations. However, since this information is unavailable, we turn towards the Generalized Grover Search (GGS) algorithm [20]. Precisely, in one GGS run, the GS quantum circuit is executed iteratively, with the number of iterations dynamically adjusted in each instance until success. In this paper, we implement the timed-out variant of GGS, with a fixed maximum budget of iterations denoted as Max_{iter}. Furthermore, in each iteration, the number of iterations j is uniformly sampled from the range $[1, l]$, where l is initially set to a constant λ. Subsequently, l is incremented by an amount λ after each iteration and is upper-bounded by $\sqrt{L}$. The rationale behind choosing λ is detailed in [23].

5.2.2 Setting Max_{iter}

Without loss of generality, in the worst case, the GS algorithm needs $\sqrt{L}$ iterations. Therefore, the iteration budget for GGS is set to be $\lfloor \alpha \cdot \sqrt{L} \rfloor$, where α is a positive rational number. A universal lower bound for α, specifically $\alpha_{lb} = 9.2$, was presented in [23]. Subsequently in the next section, we will delve into the establishment of instance-specific upper bounds, denoted as α_{ub}.

5.2.3 Boosting Success Probability

The GGS algorithm finds a valid solution with probability 0.5. To boost the probability to the desired δ, we employ the Powering Lemma [40], which resorts to repeat executions of every GGS run. The number of repetitions required for the boosting is $R = \lceil \log(\frac{1}{1-\delta}) \rceil$.

Algorithm 4 find_opt_config

Require: List: (I, W, V), Int: (W_{max}, V_{target}), Float: δ, λ, α
$R = \lceil \log(\frac{1}{1-\delta}) \rceil$
$r = 0$
$L = 2^{|I|}$
while $r < R$ **do**
  $l = \lambda$
  $Total_{iter} = 0$
  $j \leftarrow Uniform\_Sample(l)$
  while $Total_{iter} + j < \lfloor \alpha \cdot \sqrt{L} \rfloor$ **do**
    $qc \leftarrow q\_search(I, W, V, W_{max}, V_{target}, j)$
    $P_{ind} \leftarrow execute(qc)$
    $P_{res} \leftarrow validate\_pred(W, V, W_{max}, V_{target}, P_{ind})$
    if $P_{res}[qualify] == True$ **then**
      return P_{res}
    else
      $Total_{iter} = Total_{iter} + j$
      $l = \min(\lambda \cdot l, \sqrt{L})$
      $j \leftarrow Uniform\_Sample(l)$
  $r = r + 1$
return P_{res}

The above considerations are incorporated in Algorithm 4. It first determines the number of repetitions R required for the boosting the success probability of the GGS algorithm to δ. Next, for each run $r \in R$ of the GGS algorithm, the q_{search} procedure (shown in Algorithm 5) is invoked to construct a GS quantum circuit (qc) with j iterations, where j is uniformly sampled from the range $[1, l]$. Subsequently, qc is executed on a quantum platform to predict an optimal configuration. As the GS algorithm is probabilistic, the algorithm utilizes the $validate\_pred$ procedure (shown in Algorithm 6) to verify if the predicted configuration is valid. This procedure simply calculates the aggregate cost and benefit of the predicted configuration and compares it with the storage and target benefit constraints. If the validation is successful, the algorithm returns the identified configuration; otherwise, the sampling range l is adjusted, and a new iteration count j is sampled. Now, if j is within the remaining iteration budget, then the q_{search} procedure is invoked. Otherwise, the run is incremented, and l is re-initialized to λ. This whole procedure is repeated till a valid configuration is identified, or the number of runs are exhausted. At termination, the optimal configuration is identified and returned with probability δ.

Algorithm 5 provides a comprehensive overview of the construction of the Grover Search quantum circuit specifically tailored to the index selection problem. The algorithm commences by determining the required number of qubits for representing the candidate index configurations (n), as well as

their corresponding costs (w) and benefits (v). We also needs qubits to indicate individual constraints satisfaction (W_{sig} and V_{sig}, hence $c = 2$), and final candidate configuration qualification (C_{sig} hence $out = 1$). Next we initialize a quantum circuit with the needed qubit capacity along with n classical bits to store the output of the execution. Note, all the qubits are initially in the quantum $|0\rangle$ state.

Algorithm 5 q_search

Require: List: (I, W, V), Int: (W_{max}, V_{target}, j)

1: $n = |I|$ ▷ # of Index Qubits
2: $w = \lceil \log_2(\sum_{i=0}^{n-1} w_i) \rceil + 1$ ▷ # of cost qubits
3: $v = \lceil \log_2(\sum_{i=0}^{n-1} v_i) \rceil + 1$ ▷ # of Value qubits
4: $c = 2$ ▷ # of Constraint qubits
5: $out = 1$ ▷ Output qubit
6: $qc \leftarrow QuantumCircuit(n + w + v + c + out, n)$
7: $qc.h(n)$ ▷ Generate configurations (Section 5.0.1)
8: $qc.h(w + v)$ ▷ Init. cost and Value qubits
9: $qc.x(out)$
10: $qc.h(out)$ ▷ Init. out qubit in $|-\rangle$ state
11: $qc\_oracle \leftarrow config\_sel\_oracle(I, W, V, W_{max}, V_{target})$
12: $qc\_oracle\_inv = qc\_oracle.inverse()$
13: **for** $i \in range(j)$ **do**
14: $qc.append(qc\_oracle)$
15: $qc.ccx(wt_{flag}, val_{flag}, out)$
16: $qc.append(qc\_oracle\_inv)$
17: $qc.append(grover\_diffuser())$
18: $qc.measure(in_{qubit}, in_{cbit})$
19: **return** qc

Now, we generate all the candidate configurations using the quantum **H** gates as summarized in Section 5.0.1. Next, the configuration qualification qubit out is initialized in $|-\rangle$ state by applying the quantum **X** and **H** gates in succession. This is needed by the Grover Search algorithm to signal the candidate qualification using the quantum *Phase Kickback* phenomenon [6]. Next, the $config\_sel\_oracle$ procedure is invoked (described previously in Section 5.1), which generates a quantum oracle qc_oracle for our problem instance. Since the qc_oracle is fundamentally a unitary matrix, we can compute its inverse using Qiskit's inbuilt $inverse()$ procedure. In Lines 13 - 17, the quantum circuit qc is appended with j Grover's iterate operations. Each Grover's iterate comprises of an application of qc_oracle, then flagging of the out qubit followed by the Grover's diffusion operation on the index qubits. But for the diffusion operation to work as expected, we need to *un-entangle* the index qubits from other qubits which got entangled by the qc_oracle. To achieve this, we first apply the inverse qc_oracle and then Grover's diffusion operation (using the standard diffusion method available in the Qiskit library [16]). Finally, we add quantum measurement gates to measure the value of the index qubits into the classical bits and return the composed quantum circuit.

5.3 Computational Complexity

Aligned with the quantum computing literature, our complexity measure is the number of calls made to the quantum oracle, since it is architecture-independent. The SQIA _search procedure (Algorithm 3) performs recursive halving over the range $[0, V_{max}]$ which takes a maximum of $\lceil \log_2(V_{max}) \rceil$ steps.

Algorithm 6 validate_pred

Require: List: (W, V), Int: (W_{max}, V_{target}), Str: P_{ind}

$P_{res}[ind] = P_{ind}$
$P_{res}[wt] = \sum_{i=0}^{n-1} P_{ind}[i] \cdot w_i$
$P_{res}[val] = \sum_{i=0}^{n-1} P_{ind}[i] \cdot v_i$
if $P_{res}[wt] \leq W_{max}$ and $P_{res}[val] \geq V_{target}$ **then**
    $P_{res}[qualify] = True$
else
    $P_{res}[qualify] = False$
return P_{res}

Now, for every target benefit value V_{target}, the algorithm invokes the GGS algorithm $R = \lceil \log\left(\frac{1}{1-\delta}\right) \rceil$ times. In each run of GGS, we execute the GS algorithm multiple times but with a budget of Max_{iter} iteration, which is heuristically set to $\lfloor \alpha \cdot \sqrt{L} \rfloor$. Further, each iteration makes 2 invocations of the quantum oracle – once to apply the Oracle, and the second time to undo its effect. Therefore, the total number of Oracle calls made by Algorithm 3 is upper bounded by:

$$\lceil 2 \cdot \lfloor \alpha \cdot \sqrt{L} \rfloor \cdot \lceil \log\left(\frac{1}{1-\delta}\right) \rceil \cdot \lceil \log_2(V_{max}) \rceil \rceil \tag{11}$$

As mentioned earlier, a universal lower bound $\alpha_{lb} = 9.2$ for all L, and the success probability δ is set by the user during system initialization. Therefore, only the terms $\sqrt{L}$ and V_{max} vary based on the input problem instance. Further, if we reasonably assume that V_{max} can be accommodated within a 32-bit integer, then $\log_2(V_{max})$ can be upper bounded to 32. Therefore, we can conclude that the overall rate of growth of the number of oracle calls is $\mathcal{O}(\sqrt{L})$.

Further, instance-specific upper bounds, α_{ub}, are computable through Equation 11, by imposing a constraint that the resulting value must be below $\tau \cdot \sqrt{L}$, where $\tau << \sqrt{L}$. Now, depending on the index scenario, it is possible that α_{ub} may turn out to be less than α_{lb}. For such situations, it becomes impossible to simultaneously ensure the δ success probability and computational efficiency, and one has to choose α to achieve one or the other. On the other hand, if α_{ub} is greater than α_{lb}, we can set α to 9.2 to maximize the computational efficiency.

5.4 SQIA Resource Scaling

The number of qubits required for implementing the SQIA pipeline is detailed in Table 3. The entries in the table show that the qubit count exhibits a *linear* relationship with the number of indexes and a *logarithmic* correlation with the cumulative cost and benefits. However, if we reasonably assume that the aggregate cost & benefit (log terms) can fit within a 32-bit integer, in that case, the qubit count effectively scales *linearly* with the number of indexes.

6 Experiments

In principle, the correct methodology for evaluating quantum performance relative to the classical approaches would be to execute both classes of algorithms on their respective devices, assess the quality of the recommended outcomes, and measure the index selection overheads. However, this is not practical at the current time due to lack of industrial-strength quantum platforms. Therefore, we settle for

Symbol	Description	Count
n	# of Qubits for Indexes	$\|I\|$
w	# of Qubits for Cost	$\lceil \log_2(\sum_{i=0}^{n-1} w_i) \rceil + 1$
v	# of Qubits for Benefit	$\lceil \log_2(\sum_{i=0}^{n-1} v_i) \rceil + 1$
c	# of Qubits for Constraints	2
out	# of Qubits for Output	1

Table 3: SQIA Qubit Requirement

the approach prevalent in the quantum computing literature (e.g. [23, 37]), wherein comparisons are on architecture-independent metrics – in our case, the number of oracle calls.

6.1 Experiment Environment

We implemented the proposed ideas using the Qiskit SDK [16] and performed evaluations with Qiskit Aer [7], on a **32-qubit** gate-based noiseless simulator. Using Qiskit Runtime Primitives [14], the same code was also ported to and evaluated on an IBM Eagle circuit processor with **127-qubits** ("ibm_sherbrooke"). Our experiments have perforce been carried out on modest problem instances due to current platform limitations; however, we expect the design techniques to carry through to futuristic scaled platforms.

As shown in Stage 4 of Figure 2, the Quantum Index Advisor module receives an index selection problem instance comprising of (a) an index set (I), (b) the associated time benefit (V) and storage cost (W) of each index in I, and (c) the storage budget W_{max}. In our evaluation, the instance is solved with the proposed quantum schemes, OQIA and SQIA, and compared with the classical baselines, Greedy and Exhaustive Search.

Our problem suite consists of four index selection instances. The first instance, comprising 7 indexes, is the motivating example of Figure 1, generated on the commercial database engine – we refer to it as **CDB_I7**. The remaining three problem instances comprising 5, 6, and 7 indices, respectively, are synthetically generated – we hereafter refer to them as **I5**, **I6**, and **I7**. These problem instances are shown in Figure 9, and they all have the same storage constraint, namely $W_{max} = 19$. In addition, in all of them, Exhaustive Search provides the same optimal configuration, namely $\{i_0, i_1, i_2, i_3\}$ with benefit 44, while Greedy provides the same (sub-optimal) recommendation, namely $\{i_0, i_2, i_3, i_4\}$ with benefit 35. We define the quality of a configuration as its benefit normalized to the ideal solution (as obtained by the Exhaustive Search algorithm).

While the original CDB_I7 instance could be directly used with OQIA, its costs and benefits were normalized for SQIA evaluation to reduce the complexity of the quantum circuit. Specifically, the following transformed problem instance produces the same greedy and optimal solution as the original problem: $I = [i_0, i_1, i_2, i_3, i_4, i_5, i_6]$, $W = [126, 114, 3, 72, 95, 1, 4]$, $V = [4, 5, 27, 27, 27, 1, 1]$, and $W_{max} = 75$. Furthermore, the algorithmic parameters for OQIA were set to ($p = 1, S = 100$), while SQIA had ($\delta = 0.9, S = 1$). The sensitivity to these parameters is discussed later in the section.

6.2 Performance Comparison

Configuration Quality Table 4 presents a summary assessment of the configuration quality delivered by the four index selection strategies when invoked on our problem suite on the noiseless quantum simulator. For the quantum algorithms, the Weighted Average column reports the average of the quality scores over the ten repeat invocations, the Optimal Fraction column represents the fraction of outcomes delivering the optimal benefit, while the Worst Case column represents the smallest benefit

Index (I)	Benefit (V)	Cost (W)
i_0	14	7
i_1	12	6
i_2	10	4
i_3	8	2
i_4	3	1

***Storage Capacity*: 19**

(a) I5

Index (I)	Benefit (V)	Cost (W)
i_0	14	7
i_1	12	6
i_2	10	4
i_3	8	2
i_4	3	1
i_5	13	7

***Storage Capacity*: 19**

(b) I6

Index (I)	Benefit (V)	Cost (W)
i_0	14	7
i_1	12	6
i_2	10	4
i_3	8	2
i_4	3	1
i_5	13	7
i_6	12	6

***Storage Capacity*: 19**

(c) I7

Figure 9: Index Problem Suite

obtained across the invocations. Recall that the expectation from OQIA is to recommend a solution having better quality than the Greedy scheme, while SQIA should produce the optimal solution with the desired δ probability. The good news from these results is that both schemes consistently achieve their objectives (except for I5). The detailed instance-specific analysis is as follows:

CDB_I7: The Greedy configuration delivers only 0.54 of the optimal. OQIA improves the quality to 0.76, while SQIA delivers the optimal configuration with the desired $\delta = 90\%$. Nevertheless, owing to the inherently probabilistic nature of quantum platforms, the worst-case recommendation could be of arbitrary quality. However, as observed for both schemes, the worst-case quality is about the same as the greedy solution.

I5: Here, Greedy delivers a configuration quality of 0.8, and OQIA enhances the configuration quality to as high as 0.99. On the other hand, SQIA although delivering 0.9 quality, does not satisfy its desired δ success probability. This is because the α value is impractical for this instance, as explained later in the section.

I6 and I7: In both these instances, OQIA enhances the solution quality to 0.97. Further, SQIA always recommends the optimal solution, and exceeds the δ threshold. The worst-case quality of both schemes is also significantly better than the greedy recommendation. Notably, this exceptional performance is delivered despite the exponential increase in the number of candidate configurations from **I6** to **I7**.

Computational Overheads Turning our attention to the computational effort, also delineated in Table 4, we observe that for **CDB_I7**, **I6** and **I7**, SQIA incurs only marginally fewer Oracle calls compared to Exhaustive Search. This is further substantiated when we consider the smallest-sized **I5**, where the Oracle calls even exceed those of Exhaustive Search. This may seem surprising; however, this is an artifact of our small-sized examples and is again due to the impractical values of α. The resource gap will become clearly apparent in large-index scenarios seen in enterprise environments. For instance, consider the full TPC-H benchmark query suite with 53 single-attribute candidate indexes [41]. Assuming that the aggregate benefits can be accommodated in a 32-bit integer, then for $\delta = 0.9$, using Equation 11, we can estimate that SQIA will only need 0.003% of the oracle calls incurred by Exhaustive Search.

A similar trend is observed for the OQIA scheme and is attributed to the variational principle used

Problem	# of Configs	IA Scheme	Configuration Quality			Quantum Resources		Comp. Overhead
			Weighted Avg.	Optimal Fraction	Worst Case	Qubits	Depth	
CDB_I7	128	Exhaustive	1.0	1.0	1.0	–		100%
		SQIA ($\delta = 0.9$, S = 1)	**0.95**	**0.9**	0.54	28	80	73.4%
		OQIA (p = 1, S = 100)	**0.76**	0.3	0.5	15	30	22.6%
		Greedy	0.54	0	0.54	–		–
I5	32	Exhaustive	1.0	1.0	1.0	–		100%
		SQIA ($\delta = 0.9$, S = 1)	**0.9**	0.8	0.34	21	58	112.5%
		OQIA (p = 1, S = 100)	**0.99**	**0.9**	0.89	10	20	90.6%
		Greedy	0.8	0	0.8	–		–
I6	64	Exhaustive	1.0	1.0	1.0	–		100%
		SQIA ($\delta = 0.9$, S = 1)	**1.0**	**1.0**	**1.0**	22	60	93.7%
		OQIA (p = 1, S = 100)	**0.97**	0.6	0.86	11	22	45.3%
		Greedy	0.8	0	0.8	–		–
I7	128	Exhaustive	1.0	1.0	1.0	–		100%
		SQIA ($\delta = 0.9$, S = 1)	**1.0**	**1.0**	**1.0**	25	68	75%
		OQIA (p = 1, S = 100)	**0.97**	0.4	0.89	12	24	22.6%
		Greedy	0.8	0	0.8	–		–

Table 4: Evaluation of IA Schemes on a 32-qubit Quantum Simulator

by the underlying QAOA algorithm, which dynamically explores and refines the solution space, and quickly converges to optimal or near-optimal solutions. Hence, it is evident that both the proposed schemes are targeted towards larger problem instances. Later in this section, we delineate the problem size landscape in which the SQIA scheme ensures both guaranteed quality and computational advantage.

6.3 IBM Eagle Circuit Processor

We now turn our attention to the performance observed on the 127-qubit Eagle quantum circuit processor. It is one of the most performant IBM quantum systems with an average error rate of less than 2% per layered gate operation [13]. To optimize the quantum circuits for this hardware and reduce the impact of errors, we used Qiskit runtime compilation techniques and transpiled the Qiskit simulator quantum circuits for OQIA and SQIA by setting the optimization level to 3 [10] and verified that it runs correctly. Further, their performance on the index instances of Table 4 was evaluated and the results are summarized in Table 5. The Configuration Quality column reports the average of the quality scores over three repeat invocations. The outcomes of these experiments are qualitatively in agreement with the simulated results, demonstrating that achieving high-quality solutions is feasible even on these early quantum hardware. Moreover, an additional insight is that OQIA is more robust to quantum noise than SQIA, leading to better configuration quality on this platform. This is due to the QAOA algorithm used in OQIA, which is more effective in noisy hardware environments than the GS algorithm employed by SQIA.

6.4 D-Wave Leap Annealing Processor

We have also conducted experiments on the D-Wave 5000-qubit Leap annealing processor. Since annealing processors are highly suited for handling optimization problems in general, we take the QUBO generated in OQIA and supply it to the D-Wave Leap Hybrid Solver [11]. And, as expected, the annealing processor consistently provided *optimal* solutions for all the index instances considered in our study. But this exceptional success needs to be qualified with the following observations: (1) A

Problem	IA Scheme	Configuration Quality
CDB_I7	SQIA ($\delta = 0.9$, $\alpha = 0.26$, S = 1)	**0.85**
	OQIA (p = 1, S = 100)	0.68
I5	SQIA ($\delta = 0.9$, $\alpha = 0.18$, S = 1)	**0.89**
	OQIA (p = 1, S = 100)	**1.0**
I6	SQIA ($\delta = 0.9$, $\alpha = 0.22$, S = 1)	**0.86**
	OQIA (p = 1, S = 100)	**0.98**
I7	SQIA ($\delta = 0.9$, $\alpha = 0.26$, S = 1)	**0.95**
	OQIA (p = 1, S = 100)	**1.0**

Table 5: Evaluation of IA Schemes on IBM 127-qubit Eagle Quantum Circuit Processor (ibm_sherbrooke)

circuit processor of matching qubit capacity (expected within this decade as per the IBM roadmap) is likely to have similar performance quality; (2) The overall industry trend is towards circuit processors, with even D-Wave itself recently including such processors in its roadmap [1]; (3) In a practical DBMS, it appears reasonable to expect a *single* quantum platform that is usable for both generic computation as well as optimization – a circuit processor provides this computational flexibility. Therefore, hosting optimization problems on circuit processors is of independent interest.

6.5 OQIA using VQE

We have initially implemented OQIA using QAOA. The reason for our choosing QAOA over VQE (and similar techniques) to solve the QUBO problem was the expectation that these alternative techniques would incur higher computational overheads. Specifically, VQE operates over a generic solution space, whereas QAOA leverages the structure of the specific problem's solution space, and is therefore likely to be more efficient.

For completeness, we have now implemented VQE as well, and the results are shown in Table 6. In this table, the configuration quality and computational overheads of VQE are compared to QAOA for the various index problem instances. We see that although the solution quality of VQE is generally similar to that of QAOA, its computational overheads are much higher, almost by an order of magnitude.

Problem Instance	VQE Solution Quality	QAOA Solution Quality	VQE Computational Overhead wrt OQIA
CDB_I7	**0.76**	0.76	**9.0x**
I5	**0.89**	0.99	**6.5x**
I6	**0.93**	0.97	**7.3x**
I7	**0.94**	0.97	**7.3x**

Table 6: Evaluation of VQE-based OQIA

6.6 OQIA Knob Settings

Since OQIA internally uses the QAOA algorithm, there are a couple of knobs that can be set: (1) The classical optimizer used to update the parameters; (2) The repetition depth of QAOA (p); and (3) The number of shots (S). As explained below, we set the classical optimizer to COBYLA [4]. For this optimizer choice, Table 7 displays OQIA performance for various settings of the p and S knobs.

	p	Solution Quality				Q. Resource	
		S = 1	S = 10	S = 100	S = 1000	Qubits	Depth
CDB_I7	1	0.1	0.31	0.76	**1.0**	15	30
	2	0.1	0.41	0.81	**1.0**	15	46
	3	0.1	0.41	0.81	**1.0**	15	62
I5	1	0.4	0.84	**0.99**	**1.0**	10	20
	2	0.46	0.84	**1.0**	**1.0**	10	31
	3	0.53	0.88	**1.0**	**1.0**	10	42
I6	1	0.42	0.87	**0.97**	**1.0**	11	22
	2	0.45	0.88	**0.98**	**1.0**	11	34
	3	0.54	0.88	**0.98**	**1.0**	11	46
I7	1	0.49	0.84	**0.97**	**1.0**	12	24
	2	0.5	0.85	**0.98**	**1.0**	12	37
	3	0.6	0.87	**0.98**	**1.0**	12	50

Table 7: OQIA Evaluation

6.6.1 Choice of Classical Optimizer:

Qiskit offers a suite of optimizers [2]. In our experiments, we opted for the Constrained Optimization By Linear Approximation (COBYLA) optimizer. COBYLA is a gradient-free optimization algorithm that uses a linear approximation of the function in the neighborhood of the current point to determine the next point to evaluate. We chose COBYLA primarily because, being gradient-free, it does not require any additional execution of the quantum circuit to update the parameters.

6.6.2 Configuring the QAOA repetition depth p:

The depth of the QAOA quantum circuit grows proportionally with the parameter p. It is anticipated that the approximation quality will improve with p. But, there are no theoretical bounds for p – since QAOA can be seen as a Trotterized version of quantum annealing, with the adiabatic evolution employed in quantum annealing achievable in the limit as p approaches infinity. However, in practice, a common choice for p is a low value of 1, striking a balance between solution quality and quantum feasibility. Previous research, including the recent quantum join-order optimization paper [53], has consistently employed $p = 1$ in their experiments.

Notwithstanding the above practice, it has been argued in the literature that a logarithmic depth QAOA may be required to outperform classical optimizers [56]. Therefore, in our experiments, we have evaluated for $p \in \{1, 2, 3\ (= \lceil \log_2(|I|) \rceil)\}$ to conduct an empirical cost-benefit analysis. Interestingly, as p increased, we observed only a marginal improvement in the solution quality, accompanied by the expected linear rise in quantum circuit depth. The average number of optimizer calls also increased linearly, resulting in longer convergence times. This phenomenon occurs because a larger value of p supports discerning finer details within the optimization landscape. Overall, our recommendation is to utilize $p = 1$, thereby optimizing both computational efficiency and solution quality.

6.6.3 Number of Shots (S)

In our experiments, we considered $S \in \{1, 10, 100, 1000\}$. From Table 7, it is evident that, independent of the p value, the solution quality of OQIA substantively improves as the number of shots increases, and for S beyond 100, is virtually optimal. On the down side, the number of iterations also increases with the number of shots, leading to higher convergence times. This is again due to the optimizer being able to discern finer details within the optimization landscape. Given this trade-off between solution

quality and computational effort, our goal is to maximize solution quality while minimizing the number of shots.

To find the optimal number of shots, we evaluated OQIA for problems I5, I6, and I7 by setting $p = 1$ and varying S in the range $[1, 130]$. Each experiment was repeated ten times, and Figure 10 shows the average quality score against S. For comparative purposes, the performance of Greedy and Exhaustive are also shown. Three key insights emerge from this figure: 1) OQIA rapidly outperforms the greedy algorithm with a small S [≥ 10]; 2) The solution quality achieved by OQIA consistently surpasses that of the greedy solution, resulting in a superior approximation ratio; and 3) Starting from around 100 shots, the outcome is effectively optimal.

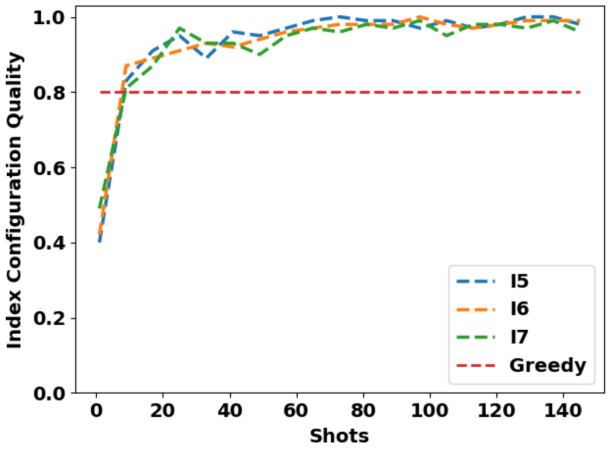


Figure 10: OQIA Sol. Quality vs Shots ($p = 1$)

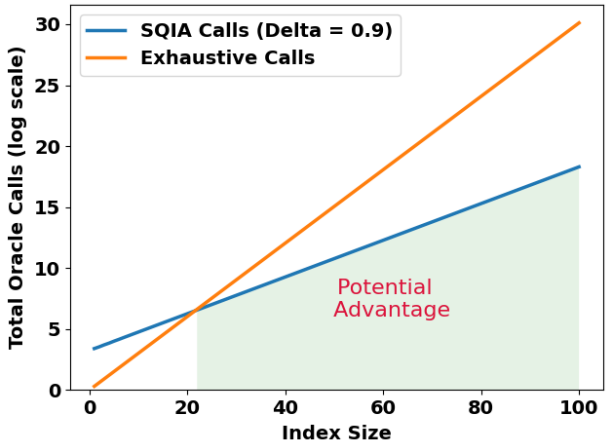


Figure 11: SQIA Advantage Space

6.7 SQIA Parameter Settings

In the SQIA scheme, we need to set the following parameters: (1) the timeout parameter (α), and (2) the number of shots (S). Moreover, there is a constraint in terms of δ, the success probability set by the user, which needs to be factored into these settings to ensure compliance. We evaluate the parameter settings for three representative values of δ: $\{0.8, 0.9, 0.99\}$ (Due to our use of the GS algorithm, an implicit lower bound on δ is 0.5).

6.7.1 Timeout parameter α:

As discussed in Section 5.3, we use Equation 11 to calculate the value of α. Due to the limited index sizes in our evaluation suite, the calculated α values were consistently lower than the minimum required for ensuring δ. For instance, for $I5$ with a $\delta \geq 0.9$, the computed value of α was so minuscule that it effectively set the iteration budget to 0. Therefore, we had no choice but to set α to the minimum feasible value, which was 0.18, such that $\lfloor \alpha \cdot \sqrt{L} \rfloor = 1$. However, this adjustment resulted in a higher number of Oracle calls as compared to the Exhaustive Search.

In other situations where $\alpha < \alpha_{lb}$, our approach aimed to enhance the likelihood of achieving an optimal configuration. This involved the introduction of a "fudge factor" into the timeout budget. The fudge factor entailed an incremental adjustment to the α value while ensuring that the SQIA scheme continued to outperform the Exhaustive Search in terms of the number of required oracle calls. As illustrated in Table 8, the outcomes indicate that for most situations, we attain the desired success probability, δ, except for a few cases highlighted in red. However, this achievement was not guaranteed

	δ	α	Solution Quality		Q. Resource	
			S = 1	S = 100	Qubits	Depth
CDB_I7	0.8	0.39	1.0	1.0	28	80
	0.9	0.26	0.9	1.0	28	80
	0.99	0.16	1.0	1.0	28	80
I5	0.8	0.2	0.8	0.8	21	58
	0.9	0.18	0.8	0.8	21	58
	0.99	0.18	0.9	1.0	21	58
I6	0.8	0.33	0.9	1.0	22	60
	0.9	0.22	1.0	1.0	22	60
	0.99	0.13	0.8	1.0	22	60
I7	0.8	0.39	0.8	1.0	25	68
	0.9	0.26	1.0	1.0	25	68
	0.99	0.16	1.0	1.0	25	68

Table 8: SQIA Evaluation

by theory. Instead, it may be attributed to an implicit side effect of the repeated executions inherent in the SQIA scheme. Notably, within the SQIA procedure, we employ a recursive halving approach, commencing with V_{max} and proceeding until the optimal target value is identified. At each step in this process, the GGS algorithm is employed to search for a valid configuration until the timeout budget is exhausted. Further, each call to the GGS algorithm is repeated R times to boost the success probability to δ. The best configuration found during all these executions is subsequently returned as the output. Consequently, even when α is set below its lower bound, this algorithmic characteristic may contribute to achieving the desired success probability, as any encounter with the optimal configuration is retained and presented as the output.

6.7.2 Number of Shots (S)

In SQIA, every GS quantum circuit (within a GGS run) is executed for S shots. After the execution, the GS algorithm uses the generated probability distribution over the observed configurations to identify and return a candidate optimal recommendation. But this comes at the cost of making additional Oracle calls incurred due to S repetition of the GS. Therefore, S should be set based on the available computational budget left after selecting the timeout parameter α. However, to explicitly demonstrate the effect of shots, we considered $S \in \{1, 100\}$ for any fixed value of δ and α. From Table 8, it is evident that for all the problem instances, the solution quality of SQIA with $S = 100$ is virtually optimal (except for I5, due to infeasible α). This is because the GS algorithm is able to identify a better solution due to repeat executions.

6.7.3 Minimum Problem Instances for SQIA

As demonstrated earlier, for smaller problem instances such as $I5 - I7$, the SQIA scheme does not theoretically guarantee the desired success probability, δ. To identify the problem sizes for which both the guarantee and the computational advantage can be expected, we set $\alpha = 9.2$ (lower bound), $\delta = 0.9$, $\log_2(V_{max}) = 32$ (assuming a 32-bit integer limit) and calculated the number of Oracle calls made by the Exhaustive Search and the SQIA scheme for the index sizes in the range $[1, 100]$. Figure 11 shows the corresponding plot with the number of oracle calls on a logarithmic scale.

In the plot, the shaded area precisely delineates the "Potential Advantage" region. Problem instances within this region demonstrate a promising potential of achieving the desired success probability (δ)

while demanding fewer Oracle calls compared to the Exhaustive Search. Specifically, for the current setting, problem instances featuring more than 22 indexes are required for viably harnessing the SQIA scheme. This is in compliance with our underlying motivation for utilizing quantum platforms, namely, *to address large problem instances* that transcend the capabilities of classical platforms.

6.7.4 Estimated Efficiency on Practical Workloads

We now project the performance profile that could be expected on the full TPC-H benchmark query suite. The number of single-attribute candidate indexes $I = 53$ [41], therefore $L = 2^{53}$. Assuming that the aggregate costs and benefits can be accommodated in 32-bit integers, the estimated depth of the Quantum Oracle circuit constructed in the SQIA scheme is around 100 (calculated by analyzing Algorithm 3). Further, as shown in [29], a single two-qubit gate currently takes around 6.5ns. Now, anticipating a reduction to 1ns within the next decade, a Quantum Oracle call in the SQIA scheme is estimated to take around 100ns. Next, assuming $\delta = 0.99$ and $\alpha = 9.2$, we use Equation 11 to estimate the number of Oracle calls made by the SQIA scheme, which is equal to $279.4 \cdot 10^9$. Multiplying this with the calculated time for one Oracle call (100ns), the SQIA scheme is estimated to take approximately 8 hours to identify an optimal configuration with 99% probability. In contrast, assuming a classical Oracle call duration of just 1ns, an Exhaustive Search would need $900.7 \cdot 10^{13}$ oracle calls and hence would take around 3.5 *months* to find the optimal solution.

7 Related Work

Recently, there have been vision papers and tutorials advocating the need to accelerate database tasks using quantum computing [24, 60, 59, 38]. But, we are not aware of any prior work performing index selection using quantum platforms. Therefore, in this section, we separately review the literature on index selection, 0-1 Knapsack Problem on quantum platform, and use of quantum platforms for database operations.

Index Selection

The index selection problem has been studied for decades, and recent comprehensive surveys are available in [39, 41]. Further, all major database engines feature an Index Advisor as an essential ancillary tool. We have already considered DB2's Index Advisor in the preceding sections. Microsoft SQL Server features a broad-based Database Engine Tuning Advisor (DTA) [27], which includes a sophisticated index advisor in its ambit. DTA considers both single and multi-column indexes, as well as their interactions.

Recently, a few machine learning (ML) based index selection methods have also been introduced as attractive alternatives to the existing strategies. A comprehensive review of these approaches can be found in [42, 43]. Most of this work uses a reinforcement learning-based approach, where the state is defined as the currently built indexes, and the action is defined as choosing an index to build. These methods exhibit promising outcomes in enhancing the index selection process's efficiency. However, as demonstrated in the detailed evaluation of [42], the quality of the recommended configuration remains similar to that of the non-ML systems.

The approaches proposed in our work aim to enhance the configuration quality provided by the above tools by leveraging the computing power offered by quantum platforms.

0-1 Knapsack Problem on quantum platform

The papers in this area could be broadly classified into two categories: Some address a weaker formulation of the original problem, while others use non-standard quantum gates with heuristic parameter settings to find close-to-optimal solutions. Specifically in [35], the authors consider 0-1 Knapsack Problem instances that do not have item-specific benefit values. Whereas in [57], an approach called "Quantum Tree Generator (QTG)" was introduced. Here, they generate in superposition all feasible solutions for a given problem instance and then leverage the Grover Search algorithm to find the solution. However, their protocol uses non-standard biased Hadamard gates and heuristically sets the bias value. These deviations raise concerns about the feasibility of implementing the proposed approaches on real quantum computers. In contrast, we have considered standard 0-1 Knapsack Problem instances and utilized only standard quantum gates in designing our solutions.

Quantum Database Platforms

There have been some earlier efforts to showcase the potential of quantum platforms for database optimization. For instance, the generation of optimal execution plans in the context of Multi-Query Optimization was proposed in the pioneering work of [54]. Their technique is based on utilizing the inherent parallelism of quantum annealing. Moreover, a singular feature of the study is its implementation on the D-Wave Quantum Annealer [5].

More recently, the core relational join-order optimization problem was addressed in [53, 46, 58] on quantum hardware. These proposals reformulated the problem to an equivalent QUBO task that can be evaluated on quantum computers. Specifically, [53] conducted a comprehensive evaluation of various query graphs and successfully generated about 41% valid join orders, among which 10% were optimal, for three-relation chain queries using the D-Wave annealing processor. These results demonstrate the feasibility of quantum solutions and also serve as a motivation for our OQIA scheme.

Another set of researchers have explored the application of quantum platforms for database transaction scheduling. Specifically, to optimally schedule transactions in a two-phase locking database [18, 19] introduced a quantum algorithm that uses quantum annealing , while [36] employs the Grover Search algorithm.

Finally, there is another line of research [49, 45] that focuses on designing "quantum-inspired" algorithms for DBMS. These algorithms are designed to run on classical computers but incorporate ideas derived from quantum computing to potentially improve their performance in solving certain problems. For instance, in [49], the authors perform resource allocation reasoning on traditional relational databases in an OLTP setting. They borrow ideas of quantum superposition and quantum measurement and allow resource transactions to commit without assigning concrete resource instances. To achieve this, they keep track of all possible worlds corresponding to all feasible concrete resource assignments.

8 Conclusions and Future Work

We presented here, for the first time, Quantum-computing-based Index Advisors for efficiently delivering index selections that provide close-to-optimal benefits under a storage budget. We first described an optimization-based approach, OQIA, which composed well-known quantum algorithms to provide high-quality configurations with limited expense of quantum resources. Then, we designed from scratch a novel Grover Search-based approach SQIA, which provides optimal solutions with high probability, in conjunction with resource consumption that is compatible with the quantum platforms expected in the coming decade. The key novelty was the construction of an efficient quantum oracle

where data is represented in qubit phases, rather than basis states, and using only standard quantum gates.

Our design is a hybrid quantum-classical architecture that lends itself to easy implementation on contemporary database environments. Using classical components to enumerate the search space and the benefits and costs of indexes, it leverages the power of the quantum computing platform for the computationally expensive index selection process.

The evaluation of modest index scenarios on both a noiseless quantum simulator and real quantum hardware demonstrated the feasibility of our proposed schemes on quantum platforms. Further, they indicated that high-quality configurations can be reliably produced by suitable choices of algorithmic parameter settings. We also showed that the complexity of our circuit design scales linearly to future deployment scenarios with large databases. In our future work, we plan to evaluate our algorithms on more powerful quantum hardware, such as the recently announced IBM Condor [12], which has 1121 qubits, and also extend our algorithms to include additional components of the IA pipeline in quantum processing.

In summary, we have taken an initial step in this paper toward designing and constructing index advisors using quantum platforms that are both close-to-optimal in solution quality and efficient with regard to index selection. We hope that our results will spur new research to address the challenges of making quantum-based databases a practical reality in the near future.

References

[1] D-wave roadmap, 2021.

[2] Qiskit optimizers, 2021.

[3] Charting the course to 100,000 qubits, 2023.

[4] Cobyla, 2023.

[5] Dwave quantum annealer, 2023.

[6] Phase kickback, 2023.

[7] Qiskit aer, 2023.

[8] Qiskit qaoa, 2023.

[9] Tpc-h benchmark, 2023.

[10] Configure runtime compilation, 2024.

[11] D-wave leap's hybrid solvers, 2024.

[12] Ibm quantum roadmap, 2024.

[13] Ibm sherbrooke 127-qubit quantum computer, 2024.

[14] Qiskit runtime primitives, 2024.

[15] S. Agrawal, S. Chaudhuri, L. Kollar, A. Marathe, V. Narasayya, and M. Syamala. Database tuning advisor for microsoft sql server 2005. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 930–932, 2005.

[16] S. Andersson et al. Learn quantum computation using qiskit, 2020.

[17] D. Basu, Q. Lin, W. Chen, H. T. Vo, Z. Yuan, P. Senellart, and S. Bressan. Cost-model oblivious database tuning with reinforcement learning. In *Database and Expert Systems Applications: 26th International Conference, DEXA 2015, Valencia, Spain, September 1-4, 2015, Proceedings, Part I 26*, pages 253–268. Springer, 2015.

[18] T. Bittner and S. Groppe. Avoiding blocking by scheduling transactions using quantum annealing. In *Proceedings of the 24th Symposium on International Database Engineering & Applications*, pages 1–10, 2020.

[19] T. Bittner and S. Groppe. Hardware accelerating the optimization of transaction schedules via quantum annealing by avoiding blocking. *Open Journal of Cloud Computing (OJCC)*, 7(1):1–21, 2020.

[20] M. Boyer, G. Brassard, P. Høyer, and A. Tapp. Tight bounds on quantum searching. *Fortschritte der Physik: Progress of Physics*, 46(4-5):493–505, 1998.

[21] N. Bruno and S. Chaudhuri. Automatic physical database tuning: A relaxation-based approach. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 227–238, 2005.

[22] N. Bruno and S. Chaudhuri. An online approach to physical design tuning. In *2007 IEEE 23rd International Conference on Data Engineering*, pages 826–835. IEEE, 2006.

[23] C. Cade, M. Folkertsma, I. Niesen, and J. Weggemans. Quantifying grover speed-ups beyond asymptotic analysis. *arXiv preprint arXiv:2203.04975*, 2022.

[24] U. Çalikyilmaz, S. Groppe, J. Groppe, T. Winker, S. Prestel, F. Shagieva, D. Arya, F. Preis, and L. Gruenwald. Opportunities for quantum acceleration of databases: Optimization of queries and transaction schedules. *Proceedings of the VLDB Endowment*, 16(9):2344–2353, 2023.

[25] S. Chaudhuri and V. Narasayya. Autoadmin "what-if" index analysis utility. *ACM SIGMOD Record*, 27(2):367–378, 1998.

[26] S. Chaudhuri and V. Narasayya. Self-tuning database systems: a decade of progress. In *Proceedings of the 33rd international conference on Very large data bases*, pages 3–14, 2007.

[27] S. Chaudhuri and V. Narasayya. Anytime algorithm of database tuning advisor for microsoft sql server, 2020.

[28] S. Chaudhuri and V. R. Narasayya. An efficient, cost-driven index selection tool for microsoft sql server. In *VLDB*, volume 97, pages 146–155. San Francisco, 1997.

[29] Y. Chew, T. Tomita, T. P. Mahesh, S. Sugawa, S. de Léséleuc, and K. Ohmori. Ultrafast energy exchange between two single rydberg atoms on a nanosecond timescale. *Nature Photonics*, 16(10):724–729, 2022.

[30] B. Dageville, D. Das, K. Dias, K. Yagoub, M. Zait, and M. Ziauddin. Automatic sql tuning in oracle 10g. In *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30*, pages 1098–1109, 2004.

[31] D. Dash, N. Polyzotis, and A. Ailamaki. Cophy: A scalable, portable, and interactive index advisor for large workloads. *arXiv preprint arXiv:1104.3214*, 2011.

[32] P. D. de la Grand'rive and J.-F. Hullo. Knapsack problem variants of qaoa for battery revenue optimisation. *arXiv preprint arXiv:1908.02210*, 2019.

[33] E. Farhi, J. Goldstone, and S. Gutmann. A quantum approximate optimization algorithm. *arXiv preprint arXiv:1411.4028*, 2014.

[34] E. Farhi, J. Goldstone, and S. Gutmann. A quantum approximate optimization algorithm applied to a bounded occurrence constraint problem, 2015.

[35] T. Fujimura. Quantum algorithm for knapsack problem by matrix computation with y-axis-rotation (-90 degrees). *Global Journal of Pure and Applied Mathematics Volume 18, Number 2, pp. 511-516*, 2022.

[36] S. Groppe and J. Groppe. Optimizing transaction schedules on universal quantum computers via code generation for grover's search algorithm. In *Proceedings of the 25th International Database Engineering & Applications Symposium*, pages 149–156, 2021.

[37] L. K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 212–219, 1996.

[38] L. Gruenwald, T. Winker, U. Çalikyilmaz, J. Groppe, and S. Groppe. Index tuning with machine learning on quantum computers for large-scale database applications. In *Quantum Data Science and Management (QDSM) (VLDB Workshops)*, 2023.

[39] S. Huang, Y. Qin, X. Zhang, Y. Tu, Z. Li, and B. Cui. Survey on performance optimization for database systems. *Science China Information Sciences*, 66(2):1–23, 2023.

[40] M. R. Jerrum, L. G. Valiant, and V. V. Vazirani. Random generation of combinatorial structures from a uniform distribution. *Theoretical computer science*, 43:169–188, 1986.

[41] J. Koßmann. *Unsupervised database optimization: efficient index selection & data dependency-driven query optimization*. PhD thesis, Universität Potsdam, 2023.

[42] J. Kossmann, A. Kastius, and R. Schlosser. Swirl: Selection of workload-aware indexes using reinforcement learning. In *EDBT*, pages 2–155, 2022.

[43] H. Lan, Z. Bao, and Y. Peng. An index advisor using deep reinforcement learning. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, pages 2105–2108, 2020.

[44] A. Lucas. Ising formulations of many np problems. *Frontiers in physics*, 2:5, 2014.

[45] S. A. Mohsin, S. M. Darwish, and A. Younes. Qiaco: a quantum dynamic cost ant system for query optimization in distributed database. *IEEE Access*, 9:15833–15846, 2021.

[46] N. Nayak, J. Rehfeld, T. Winker, B. Warnke, U. Çalıkyılmaz, and S. Groppe. Constructing optimal bushy join trees by solving qubo problems on quantum hardware and simulators. In *Proceedings of the International Workshop on Big Data in Emergent Distributed Environments (BiDEDE), Seattle, WA, USA*, 2023.

[47] M. A. Nielsen and I. Chuang. Quantum computation and quantum information, 2002.

[48] R. M. Perera, B. Oetomo, B. I. Rubinstein, and R. Borovica-Gajic. Dba bandits: Self-driving index tuning under ad-hoc, analytical workloads with safety guarantees. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*, pages 600–611. IEEE, 2021.

[49] S. Roy, L. Kot, and C. Koch. Quantum databases. In *CIDR*, 2013.

[50] S. Sahni. Approximate algorithms for the 0/1 knapsack problem. *Journal of the ACM (JACM)*, 22(1):115–124, 1975.

[51] R. Schlosser, J. Kossmann, and M. Boissier. Efficient scalable multi-attribute index selection using recursive strategies. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, pages 1238–1249. IEEE, 2019.

[52] K. Schnaitter, S. Abiteboul, T. Milo, and N. Polyzotis. Colt: continuous on-line tuning. In *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, pages 793–795, 2006.

[53] M. Schönberger, S. Scherzinger, and W. Mauerer. Ready to leap (by co-design)? join order optimisation on quantum hardware. In *SIGMOD'23: proceedings of the 2023 International Conference on Management of Data, June 18–23, 2023, Seattle, WA, USA*. ACM, 2023.

[54] I. Trummer and C. Koch. Multiple query optimization on the d-wave 2x adiabatic quantum computer. *Proceedings of the VLDB Endowment*, 9, 10 2015.

[55] G. Valentin, M. Zuliani, D. C. Zilio, G. Lohman, and A. Skelley. Db2 advisor: An optimizer smart enough to recommend its own indexes. In *Proceedings of 16th International Conference on Data Engineering (Cat. No. 00CB37073)*, pages 101–110. IEEE, 2000.

[56] J. Weidenfeller, L. C. Valor, J. Gacon, C. Tornow, L. Bello, S. Woerner, and D. J. Egger. Scaling of the quantum approximate optimization algorithm on superconducting qubit based hardware. *Quantum*, 6:870, 2022.

[57] S. Wilkening, A.-I. Lefterovici, L. Binkowski, M. Perk, S. Fekete, and T. J. Osborne. A quantum algorithm for the solution of the 0-1 knapsack problem. *arXiv preprint arXiv:2310.06623*, 2023.

[58] T. Winker, U. Çalikyilmaz, L. Gruenwald, and S. Groppe. Quantum machine learning for join order optimization using variational quantum circuits. In *Proceedings of the International Workshop on Big Data in Emergent Distributed Environments*, pages 1–7, 2023.

[59] T. Winker, S. Groppe, V. Uotila, Z. Yan, J. Lu, M. Franz, and W. Mauerer. Quantum machine learning: Foundation, new techniques, and opportunities for database research. In *Companion of the 2023 International Conference on Management of Data*, pages 45–52, 2023.

[60] G. Yuan, J. Lu, Y. Chen, S. Wu, Y. Chang, Z. Yan, T. Li, and G. Chen. Quantum computing for databases: A short survey and vision. In *Joint Workshops at 49th International Conference on Very Large Data Bases (VLDBW'23)—International Workshop on Quantum Data Science and Management (QDSM'23)*, 2023.

[61] D. C. Zilio, J. Rao, S. Lightstone, G. Lohman, A. Storm, C. Garcia-Arellano, and S. Fadden. Db2 design advisor: Integrated automatic physical database design. In *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30*, pages 1087–1097, 2004.

A Details of Problem Instances

In this section, we provide the details of the index selection problem instances used in the evaluation of OQIA and SQIA.

CDB_I5

Consider an SQL workload comprising TPC-H [9] queries Q4, Q5, Q6 and three instances of Q2 over a 1GB TPC-H database. Given this setup, the index selection problem instance generated by a popular commercial database (CDB) engine is shown in Figure 12 – we refer to it as CDB_I5. The problem instance comprises five candidate indexes, their expected time benefit wrt query response time, and storage cost overhead. Now, for a storage budget of 268MB, CDB recommends a sub-optimal index configuration $\{i_0, i_1, i_2, i_3\}$ with a total benefit of 70964, while the optimal configuration comprises indexes $\{i_1, i_3, i_4\}$ with a benefit of 194545. In this scenario, it is evident that around 64% of the available index benefit is lost due to a sub-optimal choice.

Problem Instance

Index (I)	Columns in the Index	Time Benefit (V)	Storage Cost (W)
i_0	[R_NAME, R_REGIONKEY]	17742	18
i_1	[N_REGIONKEY, N_NAME, N_NATIONKEY]	17742	1
i_2	[P_SIZE, P_PARTKEY, P_TYPE, P_MFGR]	17742	22
i_3	[PS_PARTKEY, PS_SUPPKEY, PS_SUPPLYCOST]	17742	1
i_4	[L_DISCOUNT, L_SHIPDATE, L_QUANTITY, L_EXTENDEDPRICE]	159061	266

***Storage Capacity*: 268**

CDB Recommendation	***Optimal Configuration***
Heuristic Index Set: $\{i_0, i_1, i_2, i_3\}$	*Optimal Index Set:* $\{i_1, i_3, i_4\}$
Heuristic Benefit: 70964	*Optimal Benefit: 194545*

Figure 12: Suboptimality of Heuristic IA on CDB_I5

CDB_I7

Consider an SQL workload comprising TPC-H queries Q6, Q14, Q22, and two instances of Q17 over a 1GB TPC-H database. Given this setup, the index selection problem instance generated by a popular CDB engine is shown in Figure 13 – we refer to it as CDB_I7. The problem instance comprises seven candidate indexes, their expected time benefit wrt query response time, and storage cost overhead. Now, for a storage budget of 140, CDB recommends a sub-optimal index configuration $\{i_2, i_5, i_6\}$ with a total benefit of 1302510, while the optimal configuration comprises indexes $\{i_2, i_3\}$ with a benefit of 2427540. In this scenario, it is evident that around 50% of the available index benefit is lost due to a sub-optimal choice.

I5, I6, and I7

Three problem instances comprising 5, 6, and 7 indices, respectively, are synthetically generated – we refer to them as **I5**, **I6**, and **I7**. These problem instances are shown in Figure 14, and they all have the same storage constraint, namely $W_{max} = 19$. In addition, in all of them, Exhaustive Search provides the same optimal solution: $\{i_0, i_1, i_2, i_3\}$ with benefit 44, while Greedy provides the same (sub optimal) recommendation: $\{i_0, i_2, i_3, i_4\}$ with benefit 35.

Problem Instance			
Index (I)	**Columns in the Index**	**Time Benefit (V)**	**Storage Cost (W)**
i_0	[L_DISCOUNT, L_SHIPDATE, L_QUANTITY, L_EXTENDEDPRICE]	165811	266
i_1	[L_SHIPDATE, L_DISCOUNT, L_EXTENDEDPRICE, L_PARTKEY]	178871	232
i_2	[P_CONTAINER, P_BRAND, P_PARTKEY]	1213770	8
i_3	[L_PARTKEY, L_QUANTITY]	1213770	132
i_4	[L_PARTKEY, L_QUANTITY, L_EXTENDEDPRICE]	1213770	199
i_5	[C_ACCTBAL, C_CUSTKEY, C_PHONE]	44370	2
i_6	[O_CUSTKEY]	44370	9
***Storage Capacity*: 140**			
CDB Recommendation *Heuristic Index Set:* $\{i_2, i_5, i_6\}$ *Heuristic Benefit: 1302510*		***Optimal Configuration*** *Optimal Index Set:* $\{i_2, i_3\}$ *Optimal Benefit: 2427540*	

Figure 13: Suboptimality of Heuristic IA on CDB_I7

Index (I)	Benefit (V)	Cost (W)
i_0	14	7
i_1	12	6
i_2	10	4
i_3	8	2
i_4	3	1

***Storage Capacity*: 19**

(a) I5

Index (I)	Benefit (V)	Cost (W)
i_0	14	7
i_1	12	6
i_2	10	4
i_3	8	2
i_4	3	1
i_5	13	7

***Storage Capacity*: 19**

(b) I6

Index (I)	Benefit (V)	Cost (W)
i_0	14	7
i_1	12	6
i_2	10	4
i_3	8	2
i_4	3	1
i_5	13	7
i_6	12	6

***Storage Capacity*: 19**

(c) I7

Figure 14: Index Problem Suite

I9, I15, and I20

These problem instances were constructed by extending the **I7** problem (in Figure 14c), so that for the same storage constraint, namely $W_{max} = 19$, the Exhaustive Search provides the same optimal solution: $\{i_0, i_1, i_2, i_3\}$ with benefit 44, while Greedy provides the same (sub optimal) recommendation: $\{i_0, i_2, i_3, i_4\}$ with benefit 35. Specifically, the **I9** instance is created by appending two new indexes to the **I7** instance, each with an identical cost of 7 and a benefit of 12. By adding the new indexes in this fashion, they only expand the search space but are not included in the final solution. In particular, in the Greedy ROI-based criterion, the new indexes rank low, and hence, other indexes are picked first, saturating the available capacity.

Similarly, the **I15** and **I20** problem instances are created by adding 8 and 13 new indexes to the **I7** instance, respectively, each with an identical cost of 7 and benefit of 12.