future internet

Article

QRLIT: Quantum Reinforcement Learning for Database Index Tuning

Diogo Barbosa [1], Le Gruenwald [2], Laurent d'Orazio [3], Jorge Bernardino [1,*]

[1] Polytechnic University of Coimbra, ISEC; a21280925@isec.pt; jorge@isec.pt
[2] University of Oklahoma, Oklahoma, USA; ggruenwald@ou.edu
[3] University of Rennes, CNRS, IRISA, Lannion, France; laurent.dorazio@univ-rennes.fr
* Correspondence: jorge@isec.pt

Abstract: Selecting indexes capable of reducing the cost of query processing in database systems is a challenging task, especially in large-scale applications. Quantum computing has been investigated with promising results in areas related to database management, such as query optimization, transaction scheduling and index tuning. Promising results have also been seen when reinforcement learning is applied for database tuning in classical computing. However, there is no existing research with implementation details and experiment results for index tuning that takes advantage of both quantum computing and reinforcement learning. This paper proposes a new algorithm called QRLIT that uses the power of quantum computing and reinforcement learning for database index tuning. Experiments using the database TPC-H benchmark show that QRLIT exhibits superior performance and a faster convergence compared to the classical counterpart.

Keywords: Database; Indexing; Quantum computing; Quantum reinforcement learning; Grover's search

Citation: To be added by editorial staff during production.

Academic Editor: Firstname Lastname

Received: date
Revised: date
Accepted: date
Published: date

1. Introduction

Executing queries in relational database applications with large amounts of data can take significant time. Database Management Systems (DBMS) offer various mechanisms to reduce query execution time. One such mechanism is the creation and management of indexes. Creating column indexes is a strategy that reduces the time required to search and retrieve data. However, this index selection problem, which is to find an optimal set of ~~indices~~ indexes (i.e. an optimal index configuration) for given database tables, is an NP-~~Hard~~ hard problem [1,2]. –This problem becomes more complex for large-scale database applications. Furthermore, the necessity for deleting, modifying, and inserting data may occur with considerable frequency, introducing further complexities to the problem. Indexes are managed by the database administrator (DBA), who has the knowledge about the query workload to create an efficient index configuration. As the query workload changes, the DBA must reevaluate the index configuration. To reduce the burden on the DBA, various algorithms have been proposed to automate the process of tuning database indexes in classical computing. These include algorithms that use supervised machine learning techniques to learn what indexes have been used and how queries have been performed in the past from the given training data and predict what indexes should be created for the new query workload. Since training data is often difficult to obtain, there are index tuning algorithms that make use of reinforcement learning which does not depend on training data and learns as it goes [3-16].

Quantum computing is an emerging technology that transforms the way information is processed, offering significant potential advantages over classical systems enabled by the quantum theory principles such as superposition and entanglement. This has been verified by Shor's algorithm [17] capable of factoring prime numbers in polynomial time and by a quantum search algorithm proposed by Lov Grover [18] with the ability to perform searches on unstructured data with complexity $O(\sqrt{N})$ where N is the number of elements in the search space. Additionally in database management, quantum computing has been investigated with promising results in several areas of database management, including query optimization and transaction scheduling [19] which are two other NP-Hard hard problems.

However, an analysis of the current state of the art reveals that there are no studies that implement quantum reinforcement learning strategies with experimental results in the process of automating index tuning. To address this gap, in this paper, an existing index tuning algorithm for classical computers is implemented and a quantum-classical (hybrid) version, called Quantum Reinforcement Learning for database Index Tuning (QRLIT) that employs the capabilities of Grover's search is proposed. The primary objective is to compare the performance of the hybrid algorithm against its classical counterpart.

The implemented classical algorithm [3,20] employs a machine learning technique called reinforcement learning [21]. It is composed of two principal elements, designated agent and environment. The agent learns to make decisions through interaction with the environment, using a trial-and-error learning method. The classical index tuning algorithm employs a technique called Epsilon-greedy [21] to balance the agent's ability to explore or follow its learned policy (exploiting). The proposed hybrid model replaces this technique with Grover's search algorithm, which enables a probabilistic probability approach and a natural balancing of the exploring-exploiting duality through the manipulation of the number of iterations.

Formatted: Highlight

This paper contributes with a novel algorithm that combines quantum computing with reinforcement learning to automate the process of database index tuning. Furthermore, a series of experiments demonstrate the advantages of using quantum computing over traditional system. The results obtained indicate that QRLIT converges faster to an optimal policy and is able to produce a higher reward, in terms of queries processed per hour, than its classical counterpart.

The rest of the paper is organized as follows. Section 2 provides some background information. Sections 3 presents an overview of the state-of-the-art and the classical index tuning algorithm with its quantum counterpart implementation in Section 4. The experimental environment and results obtained from running both algorithms are detailed in Section 5. Finally, Section 6 concludes the paper and proposes directions for future work.

2. Background

The purpose of this section is to provide the necessary context and foundations to understand the quantum-classical implementation. This background explains reinforcement learning, the quantum computing foundations and the Grover's quantum search algorithm.

2.1. Reinforcement Learning

In artificial intelligence, reinforcement learning is a branch inspired by the natural process of learning through reinforcement. Entities known as agents learn a policy π that maps states of the environment to actions with the purpose of maximizing the value of accumulated rewards over time in a stochastic environment modeled by a Markov Decision Process (MDP) [21]. An MDP is defined by a tuple with five elements (S, A, P, R, γ), where S represents the state space, A the action space, P the state transition function defining the dynamics of the MDP, R the reward function, and γ a discount factor with $0 \leq \gamma \leq 1$ [21].

Q-learning is a modal-free algorithm used to solve reinforcement learning problems based on temporal-difference (TD) learning methods [21]. These methods involve learning to make optimal decisions directly from experiences without a model of the environment's dynamics [21]. The core idea behind this algorithm is to learn a tabular policy, known as a Q-table, which stores the values of actions for each state. These values, called Q-values, represent the quality of each action in a specific state. In other words, it refers to how effective that action is in obtaining a good reward. So, the greater the value, the higher the potential reward, and the better the action is considered.

As a fundamental step in this algorithm, after the agent executes an action and receives feedback from the environment (reward and new state), it is crucial to update its policy. This process uses Equation 2.1, where $Q(s,a)$ represents the Q-value of the action a executed in state s, α is the learning rate, r is the reward obtained, γ is the discount rate that influences the impact of future rewards, and $max_a Q(s',a')$ represents the Q-value of the action with the highest value in the new state of the environment s'.

$$\mathrm{TDError} = r + \gamma \max_a Q(s',a') - Q(s,a)$$
$$Q(s,a) \leftarrow Q(s,a) + \alpha(\mathrm{TDError}) \tag{2.1}$$

In Q-learning, there is a limitation in balancing exploration and exploitation, as it is important to explore the environment's states to prevent the agent from getting stuck in a local maximum. To find this balance, the algorithm can use a strategy known as Epsilon-greedy [21]. This strategy uses an exploration rate epsilon ε, which decreases at the end of each episode. Therefore, with this algorithm (Equation 2.2), the agent explores with a probability of ε or follows the learned policy (exploits) with a probability of $1-\varepsilon$.

$$a = \begin{cases} \mathrm{argmax}_{a\in A}, & \text{with probability } 1-\varepsilon \\ \mathrm{random}_{a\in A}, & \text{otherwise} \end{cases} \tag{2.2}$$

2.2. Quantum Computing

Based on the principles of quantum theory, such as superposition and entanglement, quantum computing offers great advantages over classical computing [17,18]. This section is organized into two subsections that introduce and describe the building blocks of quantum computing. It begins with the introduction of the system's basic units and their mathematical representation in Section 2.2.1. Then, the quantum logic gates are introduced which are responsible for operations on the information units in Section 2.2.2.

2.2.1. Information Unit

In the field of quantum computing, the fundamental unit of information is a quantum bit, or qubit. Similarly to classical bits, qubits operate in a two-level system, corresponding to states 0 and 1. However, in contrast to bits, which exist in a single state at a time, qubits can be simultaneously in both. This phenomenon, which is paradoxical from the perspective of classical physics, is known as superposition. According to quantum theory, the precise state of a qubit in a superposition can only be identified through an observation or measurement, at which point it will collapse to one of its fundamental states, either 0 or 1, with a certain probability [22].

Mathematically, the state of a qubit is described in Dirac notation as a linear combination of the base states $|0\rangle$ and $|1\rangle$, as illustrated in Equation 2.3. The complex domain coefficients α and β represent the amplitudes of each state. The base states, designated by the symbols $|0\rangle$ and $|1\rangle$, are described in the expressions presented in Equation 2.4. The amplitudes of these states are either 0 or 1, depending on the state in question. However, in the case of superposition, the values of α and β can be included within any arbitrary value in range between $]0,1[$.

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle = \begin{bmatrix}\alpha \\ \beta\end{bmatrix} \tag{2.3}$$

$$|0\rangle = \begin{bmatrix}1 \\ 0\end{bmatrix}, |1\rangle = \begin{bmatrix}0 \\ 1\end{bmatrix} \tag{2.4}$$

A qubit can be represented on a sphere known as a Bloch sphere [22]. In this model, it is evident that the amplitudes of a quantum state are expressed in spherical coordinates, as described in Equation 2.5.

$$|\psi\rangle = \cos\frac{\theta}{2}|0\rangle + (\cos\phi + i\sin\phi)\sin\frac{\theta}{2}|1\rangle \tag{2.5}$$

Quantum state amplitudes, also known as probability amplitudes, define the probability of a superposition qubit being observed in the state $|0\rangle$ or $|1\rangle$. The probability of finding the qubit in the state $|0\rangle$ is calculated using Equation 2.6, while the probability of finding the qubit in the state $|1\rangle$ is determined by Equation 2.7.

$$P(|0\rangle) = |\alpha|^2 \tag{2.6}$$

$$P(|1\rangle) = |\beta|^2 \tag{2.7}$$

$$|\alpha|^2 + |\beta|^2 = 1 \tag{2.8}$$

In order to allow the encoding of more complex information in any computing system, it is essential to combine multiple units. In quantum computing, this combination is achieved through the tensor product of qubits. Equation 2.9 contains the notation for a system of two qubits, while Equation 2.10 presents the result of their tensor product.

$$|\psi\rangle \otimes |\omega\rangle \equiv |\psi\rangle|\omega\rangle \equiv |\psi\omega\rangle \equiv |\psi, \omega\rangle \tag{2.9}$$

$$\begin{aligned} |\psi\rangle \otimes |\omega\rangle &= (\alpha|0\rangle + \beta|1\rangle) \otimes (\gamma|0\rangle + \delta|1\rangle) = \begin{bmatrix}\alpha\\ \beta\end{bmatrix} \otimes \begin{bmatrix}\gamma\\ \delta\end{bmatrix} = \begin{bmatrix}\alpha\times\gamma\\ \alpha\times\delta\\ \beta\times\gamma\\ \beta\times\delta\end{bmatrix} \\ &= \alpha\gamma|00\rangle + \alpha\delta|01\rangle + \beta\gamma|10\rangle + \beta\delta\,|11\rangle \end{aligned} \tag{2.10}$$

2.2.2. Quantum Logic Gates

The ability to manipulate and control the amplitudes of the states of qubits is a fundamental prerequisite for the implementation of a quantum computing process. This manipulation is performed through quantum logic gates, or simply quantum gates, which allow the creation of quantum algorithms [22].

An operation is defined as a matrix that, through matrix multiplication, transforms one quantum state into another. Equation 2.11 provides a mathematical demonstration of this process, where U represents the operation in question, $|\psi1\rangle$ the initial state, and $|\psi2\rangle$ the resulting state [22].

$$U|\psi1\rangle = |\psi2\rangle \tag{2.11}$$

A quantum gate that acts on several qubits is described by a matrix of dimensions $2^n \times 2^n$, where n represents the number of qubits. The most common quantum gates are Pauli-X, Pauli-Z, Hadamard, Controlled NOT (CNOT or CX), and Controlled-Z, which are represented in matrices correspondingly in Equation 2.13. The Pauli-X gate performs state negation, which is equivalent to a NOT gate in classical computers, and Pauli-Z gate, also known as a phase-flip gate, transforms the $|1\rangle$ state into $-|1\rangle$. The Hadamard gate sets the qubit in superposition, mapping the base state as presented in Equations 2.12. The Controlled NOT is controlled by the state of a control qubit to perform the negation. In other words, the gate is activated only if the qubit is in state $|1\rangle$. In conclusion, the Controlled-Z behaves in the same way as Controlled NOT, but in this case a phase-flip operation is performed.

$$H|0\rangle = \frac{|0\rangle + |1\rangle}{\sqrt{2}}, \qquad H|1\rangle = \frac{|0\rangle - |1\rangle}{\sqrt{2}} \tag{2.12}$$

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \quad \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}, \quad \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}, \quad \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}, \quad \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix} \tag{2.13}$$

2.3. Grover's Search Algorithm

Quantum computing can speed up various search processes on unordered data due to the ability to superposition quantum states, thus allowing the use of quantum parallelism. In 1996, a search algorithm that uses these quantum properties was proposed by Lov Grover [18]. Grover's algorithm evaluates whether a given solution, called "good state", is contained in the domain of N possible solutions. By increasing the probability of the "good state" and reducing the probability of the remaining ones, it allows search with a time complexity $O(\sqrt{N})$, presenting a great advantage in relation to the classical one with a time complexity $O(N)$.

The algorithm is built with three main layers (Figure 2.1), each encapsulating a different function. The initial layer, designated as State Preparation, initiates the process by placing all qubits into a superposition state. The Oracle, representing the second layer, encodes the "good state" and changes its signal (phase shift) through a combination between Multiple Controlled Pauli-Z and Pauli-X gates [23]. Finally, the Amplification Layer, or Diffusion Operator, serves as a third layer and uses a combination of Hadamard, Multi Controlled Pauli-X, and Pauli-X gates [23]. Its function is to phase shift again and amplify the probability of obtaining the "good state" during the observation process.

Following Grover's definition, to achieve the maximum probability of measuring the good state, we need to add more iterations by repeating the layers two and three by t times (Equation 2.14) for a unique solution [24].

$$\mathrm{T} = \mathrm{int}(\frac{\pi}{4}\sqrt{\mathrm{N}} - \frac{1}{2}), \qquad \mathrm{N} = 2^{\text{number of qubits}} \tag{2.14}$$

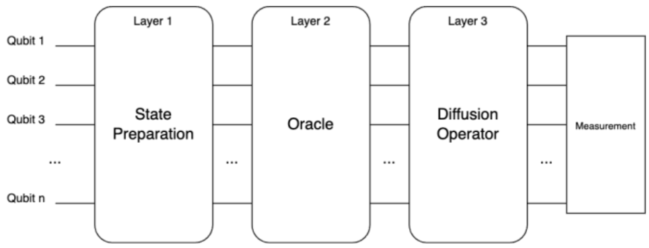

Figure 2.1. Circuit diagram of Grover's algorithm layers (based on [24]).

3. Related Work

This section presents an overview of the current state of the art for classical index tuning algorithms that employ reinforcement learning and quantum index tuning algorithms. It concludes with a detailed description of the selected classical algorithm to be converted into a quantum version.

3.1. Classical index tuning algorithms using reinforcement learning

Nowadays, there has been a significant contribution in the domain of index tuning, which plays a fundamental role in the efficacy of database searches.

In [25], the authors synthesize the current state-of-art of this subject, referring various index optimization methods, including methods using Reinforcement Learning. The method COREIL [4] uses policy iteration as algorithm, while SMARTIX [3] uses linear Q-Learning. As an evolution of SMARTIX, the authors in [20] present an approach that corrects the implementation of the TPC-H benchmark [26], which involved the execution order of the queries being processed incorrectly, as it differed from that specified in the TPC-H documentation [27]; The methods NoDBA [5], Lan's DQN [6], DRLindex [7], MANTIS [8] and DRLISA [9] implement Deep Q-Networks (DQN) as algorithm; Welborn's index advisor [10] uses Sinkhorn Policy Gradient while SWIRL [11] the Proximal Policy Optimization (PPO); BAIT [12] and AutoIndex [13] adopt Monte Carlo Tree Search (MCTS) and Lai's PPO-MC [14] use Proximal Policy Optimization-Monte Carlo (PPO-MC); Finally, DBABandit [15] and HMAB [16] use a technique called de Multi-Armed Bandit (MAB) as an algorithm.

The methods presented use a variety of approaches to solve the problem of automating indexes, however they were designed for classical computers.

3.2. Quantum algorithms for index tuning

There exists little research in the area of quantum index tuning. The article [2] proposes the conversion of the classical algorithm DINA (Deep Reinforcement Divergent Index Advisor) [28] to a quantum version. However, the paper is at an early stage; it has not provided quantum implementation details and experimental results.

Besides the capabilities that reinforcement learning provides to automate the index tuning problems, other techniques are used. The paper [29] leverages the capabilities of quantum annealers by proposing novel techniques to map the database indexes into the qubits of the quantum annealer. One technique exploits the qubits more efficiently by reducing the asymptotic qubit growth from quadratic to linear by incorporating additional auxiliary variables. The second technique is embedded within the transformation function, where efficiency is achieved through a process of extensive pre-processing before the run time. This technique generates a library of embedding templates which cover a subset of index selection problem instances.

The paper in [30] proposes SQIA, a quantum-classical (hybrid) index advisor that delivers optimal solutions with high probability by using a novel Grover Search-based approach. This approach implements an efficient quantum oracle used in the Grover search algorithm which loads the problem dada into the qubit phases. In other words, this technique loads and encodes the storage cost, benefits, and constraints.

The present literature review reveals that, in addition to the vision paper proposing a quantum counterpart of DINA [21], there is currently no quantum counterpart implementation with experimental results of index advisory using reinforcement learning.

3.3. The Classical SMARTIX Algorithm

The SMARTIX experiments presented by the authors [3] demonstrated a good balance between the disk space utilized by its index configuration and the performance metric it can achieve, which led to the selection of its evolution [20] as the foundation for the development of a quantum version in our work. As the authors of [20] have made the source code publicly available on GitHub [31], our work is built on that code, containing the adaptations required to fit the quantum algorithm and preserving the original characteristics. For the environment, they utilize a scalable database benchmark, TPC-H [26], which offers a set of features. These features allow the generation of data for a predefined group of database tables and the construction of 22 instances of queries according to 22 query templates.

The TPC-H benchmark schema includes eight database tables, each with a distinct set of attributes. When these attributes are added together, the state space contains 45 of these being available for indexing. Each attribute has two possible actions (CREATE or

DROP), which generate a state space with a total of 90 actions, each encoded in a natural decimal value in the interval [0, 89].

The reward is defined by a TPC-H performance metric, expressed in queries-per-hour (QphH) (Equation 3.3), which is composed of two other metrics: Power and Throughput. The Power metric is designed to measure the computing speed of simple queries (Equation 3.2). The Throughput metric measures the capacity to process the maximum number of queries in the shortest time using parallelism mechanisms (Equation 3.1).

The equations are composed of several elements. The quantity 3600 represents the number of seconds per hour, while the variable $QI(i,0)$ denotes the execution time of query i. The variable $RI(j,0)$ symbolizes the execution time of the refresh function j, which is responsible for inserting and removing records from the database. The variable S represents the number of query streams executed, SF the scale factor of the database, T_s the total time needed to run the throughput test for the S streams, and finally, $@Size$ represents the size of the database.

$$\mathrm{Throughput@Size} = \frac{\mathrm{S} \times 22}{\mathrm{T_s}} \times 3600 \times \mathrm{SF} \tag{3.1}$$

$$\mathrm{Power@Size} = \frac{3600}{\sqrt[24]{\prod_{i=1}^{22} \mathrm{QI}(i,0) \times \prod_{i=1}^{2} \mathrm{RI}(j,0)}} \times \mathrm{SF} \tag{3.2}$$

$$\mathrm{QphH@Size} = \sqrt{\mathrm{Power@Size} \times \mathrm{Throughput@Size}} \tag{3.3}$$

To address the issue of a tabular policy, SMARTIX uses a variant of Q-learning, called Q-learning with linear feature approximation, as its reinforcement learning algorithm [32]. This policy is represented by a set of weights, collectively referred to as a feature vector. A feature is defined as an element of the state space or the action space, so the vector has a total of 135 weights, with an additional weight corresponding to a bias.

To calculate the Q-value, Equation 3.4 must be used, where θ is the weight value and $f_n(s)$ is the value of each feature according to the current state of the environment.

$$\widehat{Q}(a,s) \leftarrow \theta_0 + \theta_1 f_1(s) + \theta_2 f_2(s) + \ldots + \theta_n f_n(s) \tag{3.4}$$

However, during the learning process, it is crucial to modify the agent's policy. The algorithm uses the temporal difference strategy with gradient descent (Equation 3.5).

$$\theta_i \leftarrow \theta_i + \alpha(r + \gamma \max_a \widehat{Q}_\theta(s',a') - \widehat{Q}_\theta(s,a)) \frac{\partial \widehat{Q}_\theta(s',a')}{\partial \theta_i} \tag{3.5}$$

The SMARTIX algorithm works as follows: ~~Initially~~First, the feature vector is populated with random values, and the replay memory is set to an empty state. ~~Secondly~~Second, a cycle is initiated, based on a predefined number of episodes. In each episode, the database is set to an initial state s. Subsequently, a sequence of steps is initiated. In each step, the algorithm determines the action to be executed in the environment using the Epsilon-greedy strategy and executes that action. Then, the environment moves to the new state and returns the reward r (QphH) and its new state s'. With the reward obtained, the algorithm updates the feature vector. The algorithm then stores the experience and selects a mini batch of experiences, and runs ~~running~~ a replay on this data. Finally, the new state becomes the current state, and the algorithm repeats the sequence of steps for each episode until the episodes reach the end.

Formatted: Highlight

4. QRLIT: The Quantum-Classical Implementation of the Classical SMARTIX Algorithm

This section describes the implementation of our QRLIT algorithm, a hybrid quantum-classical version of SMARTIX. Initially, we present a method for combining Quantum Computing (QC) with Reinforcement Learning (RL), which serve as the basis for the development of QRLIT. Then, we provide a description of the process used to identify the components that were converted. Finally, we demonstrate and ~~calculate~~ compute the fundamental values necessary ~~essential~~ for the construction of the quantum circuit, ~~concluding~~ and conclude with the QRLIT flow diagram and pseudocode.

Formatted: Highlight

Quantum Reinforcement Learning (QRL) is a method that combines the capabilities of QC and RL. Similar to the classical counterpart, Quantum Reinforcement Learning also includes a policy, a state space, an action space, and a reward function, but is inspired by the superposition principle and quantum parallelism [33]. Based on the novel algorithm proposed in [33] for QRL, the authors of the paper [34], propose an algorithm called Quantum Q-Learning (QQRL) that stores the policy in a superposition state and uses the Grover's algorithm as a strategy to amplify the probability amplitude of the best action, based on the learned policy. Grover's algorithm exploits the natural behavior of superposition states and offers a good balance between exploration and exploitation. This balance can be achieved by controlling the number of Grover iterations L through the learning process of the agent. In other words, as the agent learns and the number of iterations increases, the capacity to explore decreases until reaching the number of iterations t (as defined in Equation 2.14), which maximizes the probability of measuring the "good" action. The number of iterations L is determined by the formula in Equation 4.1 from [34], where k represents a rate that controls the proportion of policy and reward contributions, and t denotes the maximum number of possible iterations.

$$L = \min\left(\mathrm{int}\left(k\left(r + \max_{a'} Q(s', a')\right)\right), t\right) \quad (4.1)$$

Our QRLIT implementation is based on the QQRL algorithm. Therefore, we identified that the Epsilon-greedy procedure is replaced by the Grover search algorithm, keeping the remaining elements in a classical system. With Grover's algorithm in QRLIT, we are capable not only to determine the actions to be executed in the environment~~,~~ but also to naturally balance the agent's duality between exploration and exploitation. As previously outlined in the Background section, the Grover's algorithm contains three distinct layers: State Preparation, Oracle, and Amplitude Amplification. In the State Preparation layer, we initiate the policy of the agent in a superposition state and in the Oracle, we encode the action with the highest Q-value in the current state of the environment. As the last layer, we implement the Amplitude Amplification which amplifies the probability amplitude to measure the action encoded in the Oracle.

To run and build the Grover's algorithm, it is crucial to identify how many qubits are required in the quantum register to encode the actions. We calculate the number of qubits by using the formula $N_a \leq 2^n \leq 2N_a$ presented by the authors of the paper [33]. In this formula, N_a represents the size of the action space, while n denotes the number of qubits required to encode an action. We apply and solve the formula for a space of 90 actions and round off the excess, so that n is equal to ~~rounded in excess, resulting in n to be equal to~~ 7 (Equation 4.2). We then ~~Then, we~~ define the maximum number of Grover iterations, t. As the number of qubits is already calculated, N is equal to 128, and therefore t equals to 8 (Equation 4.3).

Formatted: Highlight

Formatted: Highlight

Formatted: Highlight

$$2^n = N_a \equiv n = \log_2(N_a) \equiv n = \log_2(90) \equiv n \approx 6.491 \approx 7 \quad (4.2)$$

$$N = 2^{\text{number of qubits}} \equiv N = 2^7 = 128$$

$$t = \mathrm{int}\left(\frac{\pi}{4}\sqrt{N} - \frac{1}{2}\right) \equiv t = \mathrm{int}\left(\frac{\sqrt{128}\pi}{4} - \frac{1}{2}\right) \equiv t = \mathrm{int}(8.386) = 8 \quad (4.3)$$

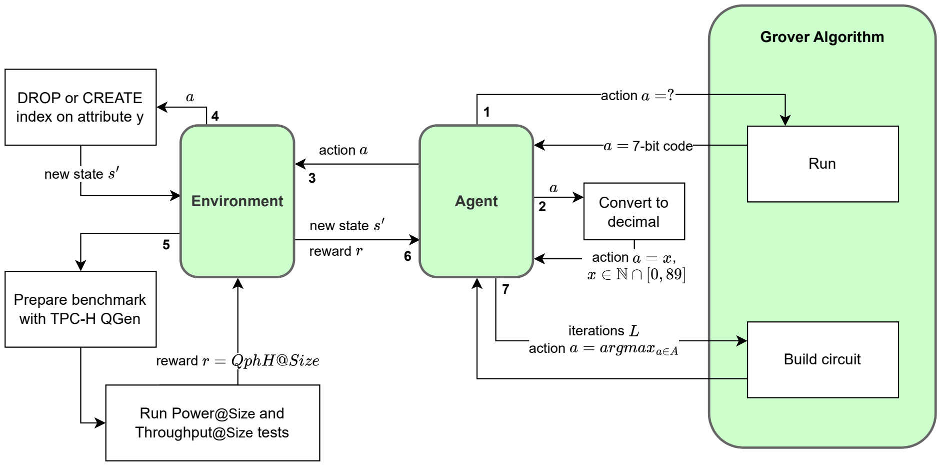

Figure 4.1. Flow diagram of the QRLIT algorithm

As identified before, to create the QRLIT, we replace the Epsilon-greedy strategy in the classical algorithm with the Grover's search (~~Line~~ line 6 in Algorithm 1). Figure 4.1 illustrates the interactions between the principal components of QRLIT. The agent component initiates the first interaction through the execution of Grover's algorithm, which returns the action a. In binary code, the action is converted to a decimal value and executed in the environment (~~Line~~ line 7 in Algorithm 1). The environment then processes the value and ~~transitions~~ enters to a new state s'. In this new state, the benchmark is prepared by creating the necessary query instances using QGen to run ~~the required query instances with QGen for the execution of~~ the power and throughput tests. Once the benchmark has been executed, the reward r and the new state s' of the environment are returned to the agent. With these two values, the agent calculates the number of Grover iterations L, selects the action of the new state that contains the highest Q-value and sends these values to the operation that builds the Grover algorithm circuit (~~Line~~ line 8 in Algorithm 1). Then, the quantum circuit is constructed with all seven qubits initialized in the register in the state $|0\rangle$. Our QRLIT proposal ~~offers~~ provides a natural balance between the exploration and exploitation, allowing for ~~thus enabling~~ a more effective learning; as the agent learns and adjusts its policy, the exploration rate decreases (Equation 4.1). Furthermore, given the properties of quantum parallelism ~~paralleling~~ and superposition states in Grover's algorithm, this proposal ~~provides~~ offers another advantage: it is able to find an action faster (complexity of $O(\sqrt{N})$) (~~Line~~ line 6 in Algorithm 1) than its classical counterpart (complexity of $O(N)$).

Formatted: Highlight

Formatted: Highlight

Formatted: Highlight

Formatted: Highlight

Algorithm 1 QRLIT algorithm with Grover's search, function approximation and experience replay. Adapted from [3] and [34].

1: Random initialization of parameters θ
2: Empty initialization of replay memory D
3: **for** each episode **do**
4: $s \leftarrow$ *DB initial index configuration mapped as initial state*
5: **for** each step of episode **do**
6: $a \leftarrow$ Run Grover algorithm on s
7: $s', r \leftarrow execute\ (a)$

Build Grovers circuit with L and $argmax_{a \in A}$
for $\theta_i \in \Theta$ **do**
Update θ_i according to Equation 3.5
end for
Store experience $e = \langle s, a, r, s' \rangle$ in D
Sample random mini-batch of experiences $e \sim D$
Performance experience replay using sampled data
$s \leftarrow s'$
end for
end for

5. Performance Evaluation

This section presents the experiments conducted on the classical algorithm SMARTIX and its quantum-classical version QRLIT (source code in [35]), and the analyses performed on the results. It is organized into three subsections: Subsection 5.1 provides a detailed description of the environment used to execute the experiments, and Subsections 5.2 and 5.3 presents the experiment results and their analysis.

5.1. Experimental Model

All experiments were conducted on a docker container with Ubuntu 22.04 in a 2021 MacBook Pro, which is equipped with 16GB of RAM, 1TB of disk space, and an Apple M1 Pro CPU with 10 cores. MySQL was used as DBMS, which implements the TPC-H benchmark, while a simulator provided by the Qiskit SDK was used to build and execute the quantum algorithm.

Additionally, in accordance to the TPC-H benchmark specification [27], 22 query instances were executed in the Power metric and 44 in the Throughput metric with 2 parallel streams (22 queries for each stream). This resulted in a total of 66 query instances being executed in each time step. The queries were generated through a tool provided by the TPC-H benchmark, designated as QGen.

The experiments were carried out according to the parameter settings outlined in Table 5.1. The first parameter setting corresponds to the tests conducted in Subsection 5.2 to study the overall performance of the algorithms when the database size is fixed at 10 MB, while the second parameter setting corresponds to Subsection 5.3 to study the impact of the database sizes of 10 MB, 20 MB, 30 MB, 40 MB, 70 MB and 100 MB on the performance of the algorithms. In this second configuration, the number of episodes was reduced to 25 in order to reduce the time required to execute the experiments.

Table 5.1. Configuration parameters for the tests.

Test Name	Database size	α	γ	k	Episodes	Steps	Total time steps
Overall Performance	10 MB	0.001	0.8	0.00017	50	100	5000
Impact of Database Size	10 MB, 20 MB, 30 MB, 40 MB, 70 MB, 100 MB	0.001	0.8	0.00017	25	100	2500

5.2. Overall Performance

In this section, experiments were conducted to study the overall performance of the two algorithms when the database is fixed at 10 MB. This study is based on the following metrics: number of queries processed per hour, episode execution time, temporal difference error, and number of Grover iterations. The first metric defines the quality of the algorithms in terms of their ability to identify a policy that maximizes the cumulative reward (queries per hour) over time. The episode execution time metric measures the velocity of the algorithms in executing an episode. The temporal difference error (TD Error) (Equation 2.1) metric demonstrates the algorithm's convergence to an optimal policy, in other words, the closer the values are to 0, the better the policy is. Finally, the Grover iterations metric measures the relation between exploration and exploitation: the lower the value, the higher the rate of exploration relative to exploitation. This metric allows the analysis of the agent's exploration capacity, which is directly correlated with the number of iterations.

The results obtained for each metric in each of 50 episodes for the two algorithms are shown in Figures 5.1-5.4. The average results of each metric over 50 episodes of the two algorithms are summarized in Table 5.2. The analysis of Figure 5.1 and Table 5.2 reveals that on average, the hybrid algorithm exhibits a higher number of queries processed per hour by 0.61% compared to its classical counterpart.

Table 5.2. Comparison results of the average results of the Classical and Quantum-Classical algorithms for the database size of 10 MB.

Metric	Classical	Quantum-Classical	Increase in Quantum-Classical over Classical
Average Number of Queries Processed Per Hour (QphH)	735,715.44	740,267.11	0.61%
Average Episode Execution Time (Seconds)	77.58	99.06	21.67%
Average Temporal Difference Error	-605.28	13.00	N/A
Average Number of Grover Iterations	N/A	7.53	N/A

Besides that, from the analysis of Figure 5.2, the hybrid algorithm has a much faster convergence to a low temporal difference error showing a more stable learning, displaying a temporal difference error trajectory closer to 0 (Table 5.2).

To find a balance between exploring and exploiting, the classical algorithm implements a strategy known as Epsilon-greedy. This strategy uses an exploration rate epsilon $\varepsilon = 0.9$, which decreases with an exploration discount factor of 0.1 at the end of each episode. Therefore, with this algorithm, the agent explores with a probability of ε or follows the learned policy (exploits) with a probability of $1 - \varepsilon$. In the case of the quantum-classical algorithm, as the agent learns and adjusts its policy, the number of Grover Iterations also increases, consequently reducing the exploration probability (Equation 4.1) (Figure 5.3).

The quantum-classical algorithm provides a better index recommendation resulting in a higher number of queries processed per hour than the classical algorithm because, as the agent of the quantum-classical algorithm refines its policy through learning, the exploration rate decreases. This leads to the decrease of unnecessary explorations and allows for more effective learning. However, the quantum-classical algorithm takes on average 21.67% more time to complete an episode than its classical counterpart (Figure 5.4). This discrepancy is related to the additional computational overhead to create and execute the quantum circuit at each time step. The Figure 5.5 show the time required to build and execute the Grover's Algorithm.

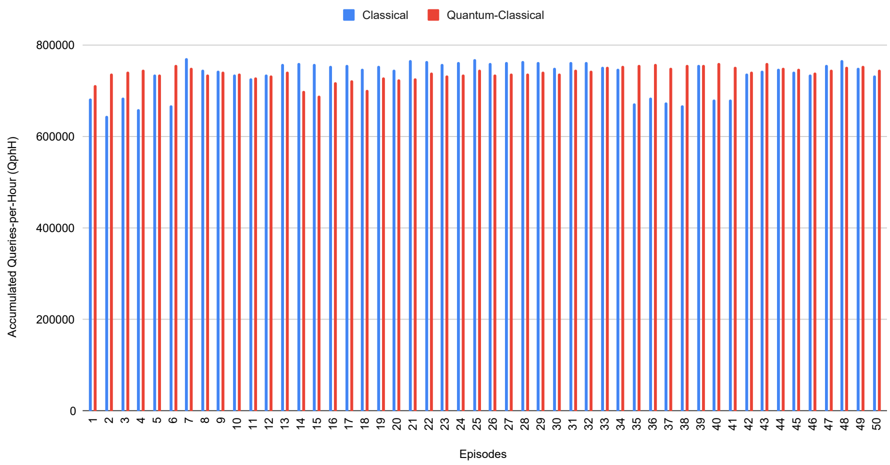

Figure 5.1. Number of queries processed per hour in each episode.

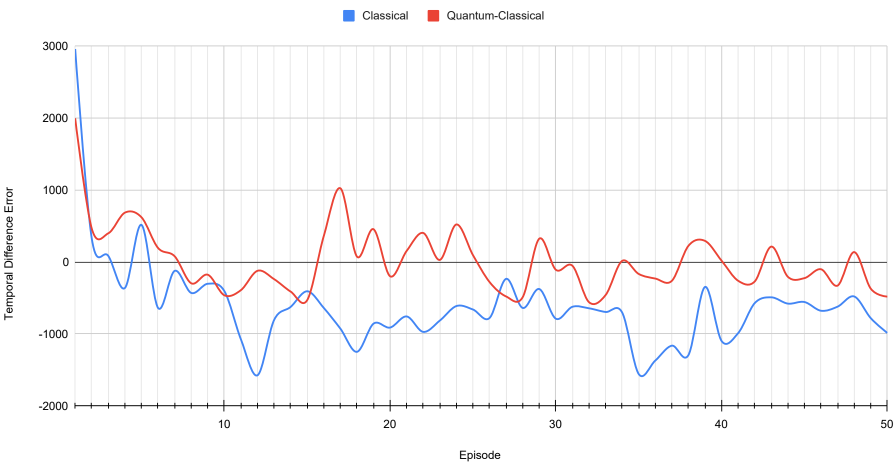

Figure 5.2. Average temporal difference error in each episode.

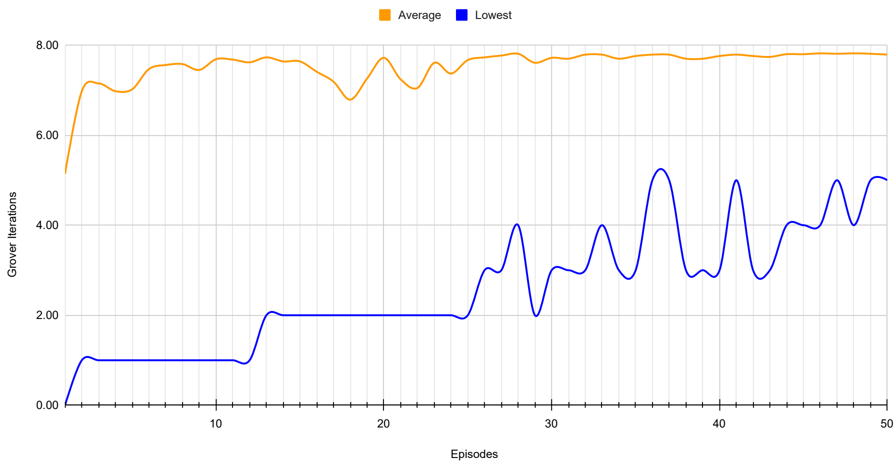

Figure 5.3. Grover iterations in each episode on hybrid algorithm.

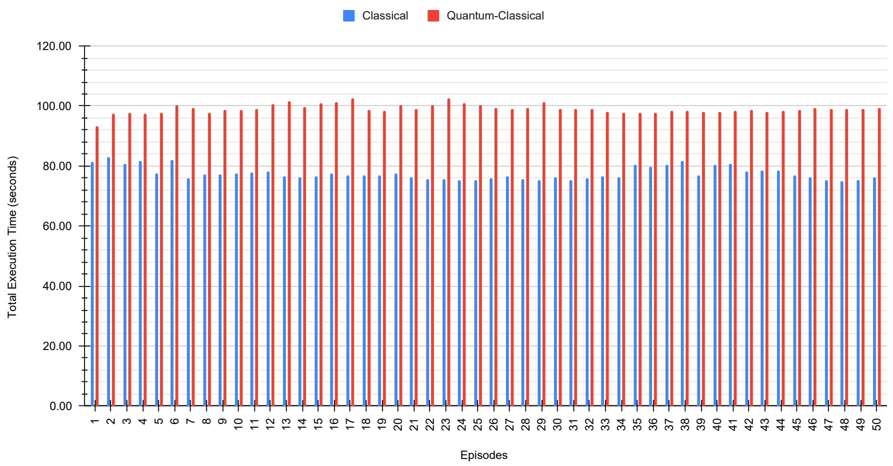

Figure 5.4. Total execution time in each episode.

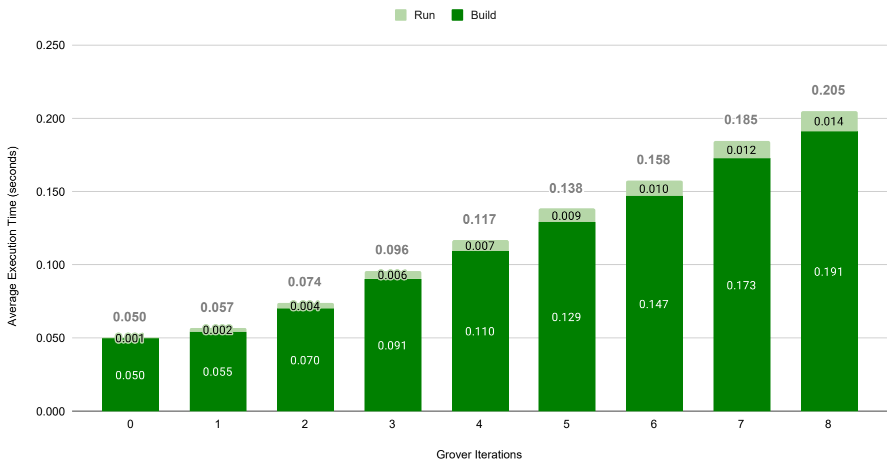

Figure 5.5. Average execution time to build and run the Grover Algorithm for each Grover Iteration in 20 executions in the quantum-classical algorithm.

5.3. Impact of Database Sizes

The purpose of this section is to examine the behavior of the algorithms across a range of database sizes, specifically 10 MB, 20 MB, 30 MB, 40 MB, 70 MB, and 100 MB. The metrics employed in this analysis include the average number of queries processed per hour of the 25 episodes for each database size, the number of Grover iterations, and the temporal difference error of each algorithm.

The results obtained for each metric in each database size for the two algorithms are shown in Figures 5.6-5.10. The average results of each metric over the database sizes of the two algorithms are summarized in Table 5.3. The analysis for the database sizes also indicates a superiority of the hybrid algorithm. The results in Table 5.3 and Figure 5.6 show that, on average, the hybrid algorithm yields a higher number of queries processed per hour of 2.49% compared to its classical counterpart and displays a temporal difference error trajectory closer to 0 (Figure 5.7). This trajectory is more evident in this analysis because the number of episodes is reduced by half, highlighting the importance of a faster convergence.

Table 5.3. Comparison results of the classical algorithm and the quantum-classical algorithm with different database sizes.

Metric	Classical	Quantum-Classical	Increase in Quantum-Classical over Classical
Average Number of Queries Processed Per Hour (QphH)	607,650.60	623,136.44	2.49%
Average Database Size Test Execution Time (Seconds)	8,449.18	8,936.49	5.45%
Average Temporal Difference Error	-603.56	13.78	N/A
Average Number of Grover Iterations	N/A	6.27	N/A

Furthermore, it can also be observed that the number of queries processed per hour decreases as the database size increases. This indicates that the size of the database affects the number of QphH generated, in other words, the reward. Consequently, according to Equation 4.1, which calculates the number of Grover's iterations and since the Q-values are directly related to the reward, they will also have smaller values. Thus, the smaller the policy and reward contribution, the smaller the number of iterations (Figure 5.8), which increases the exploration rate (Figure 5.9). Excessive exploration causes the agent not to follow the learned policy, resulting in a mostly random configuration of indexes as the database size increases.

In conclusion, besides the average superiority verified by the quantum-classical algorithm, the results in Figure 5.8 also demonstrates the need to adjust the parameter k, which regulates the reward and policy contributions to the number of Grover's iterations. In this case, as the reward value decreases, it is necessary to increase the value of k to reduce the exploration rate.

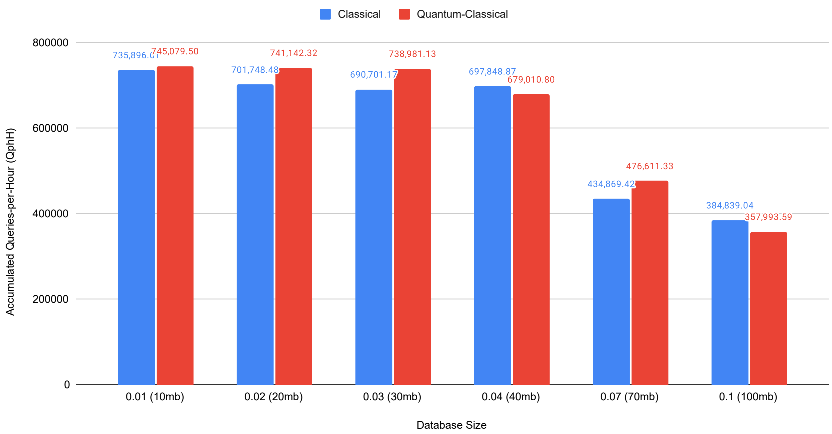

Figure 5.6. Impacts of database size on average number of queries processed per hour.

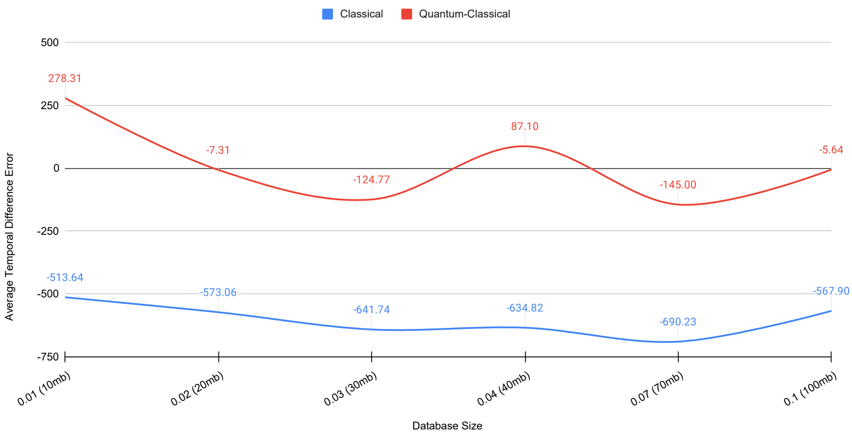

Figure 5.7. Impacts of database size on average temporal difference error.

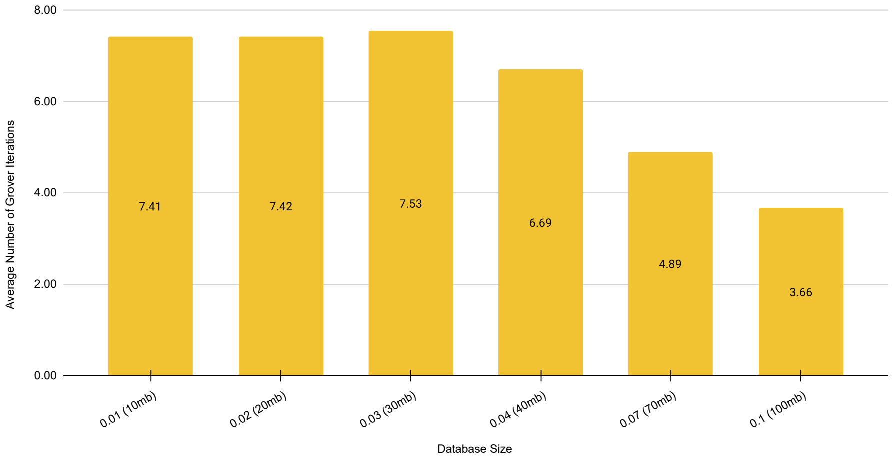

Figure 5.8. Impacts of database size on average number of Grover iterations in the quantum-classical algorithm.

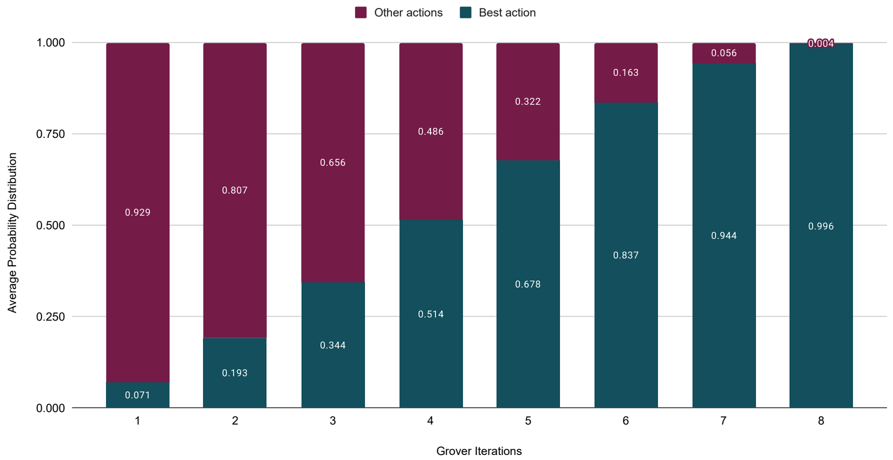

Figure 5.9. Average probability distribution of actions in 1024 shots for each Grover iteration in 10 executions in the quantum-classical algorithm.

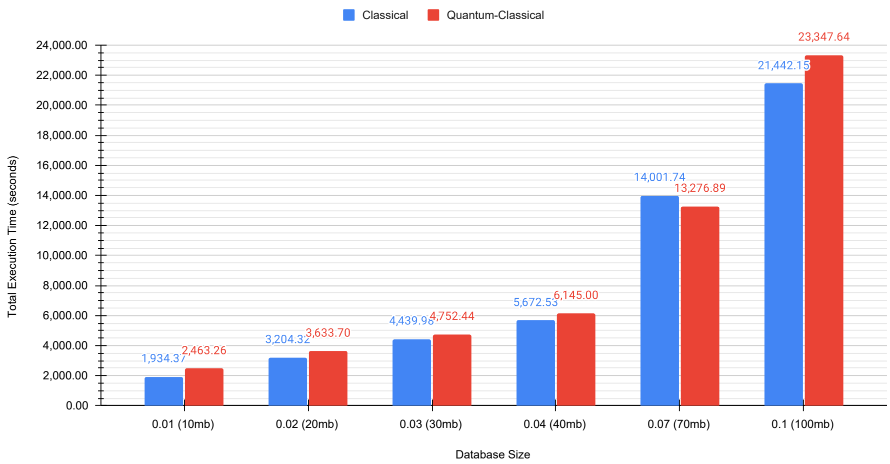

Figure 5.10. Impact of database size on total execution time.

6. Conclusions and Future Work

This work presents the implementation of QRLIT, a hybrid quantum-classical version of SMARTIX [3]. The QRLIT demonstrated better performance than the classical counterpart in terms of number of queries processed per hour and a faster convergence to an optimal policy. By controlling the Grover iterations through the reward and the agent policy, as the agent refines its policy through learning, the exploration rate decreases, allowing for a superior temporal difference error convergence closer to zero with a more effective learning compared to its classical counterpart. However, as the value of k controls the contribution of reward and policy to the number of Grover iterations, the increase in database size reveals the necessity to adjust this parameter manually to balance the exploration rate. This manual adjustment in an automatic system is a limitation because the reward (QphH) varies not only according to the size of the database, but also according to the quality and capacity of the machine's hardware.

As future work, we intend to analyze the behavior of the algorithms in databases with significant sizes and more queries. It would also be important to investigate their performance on distributed database systems. Finally, evaluating the execution of the quantum-classical algorithm on a real quantum computer is another direction for future research.

Acknowledgement

This work is supported in part by the National Science Foundation under Grant No. 2425838.

References

1. M. P. Consens, K. Ioannidou, J. LeFevre and N. Polyzotis, "Divergent physical design tuning for replicated databases," in *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, 49-60, p. 2012.
2. 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 *Joint Proceedings of Workshops at the 49th International Conference on Very Large Data Bases (VLDB 2023), Vancouver, Canada, August 28 - September 1, 2023*, CEUR-WS.org, 2023.

3. G. Paludo Licks, J. Colleoni Couto, et al., "SmartIX: A database indexing agent based on reinforcement learning," *Applied Intelligence,* vol. 50, pp. 2575-2588, 2020.
4. Basu, Debabrota et al., "Cost-Model Oblivious Database Tuning with Reinforcement Learning," in *Database and Expert Systems Applications*, Cham, Springer International Publishing, 2015, pp. 253-268.
5. A. Sharma, F. M. Schuhknecht and J. Dittrich, *The Case for Automatic Database Administration using Deep Reinforcement Learning,* 2018.
6. 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*, New York, Association for Computing Machinery, 2020, p. 2105–2108.
7. Z. Sadri, L. Gruenwald and E. Lead, "DRLindex: deep reinforcement learning index advisor for a cluster database," in *Proceedings of the 24th Symposium on International Database Engineering & Applications*, New York, Association for Computing Machinery, 2020, pp. 1-8.
8. V. Sharma, C. Dyreson and N. Flann, "MANTIS: Multiple Type and Attribute Index Selection using Deep Reinforcement Learning," in *Proceedings of the 25th International Database Engineering & Applications Symposium*, New York, Association for Computing Machinery, 2021, p. 56–64.
9. Y. Yan, S. Yao, H. Wang and M. Gao, "Index selection for NoSQL database with deep reinforcement learning," *Information Sciences,* vol. 561, pp. 20-30, 2021.
10. J. Welborn, M. Schaarschmidt and E. Yoneki, *Learning Index Selection with Structured Action Spaces,* 2019.
11. J. Kossmann, A. Kastius and R. Schlosser, "SWIRL: Selection of Workload-aware Indexes using Reinforcement Learning.," in *EDBT*, 2022, pp. 2-155.
12. W. Wu, C. Wang, T. Siddiqui, J. Wang, V. Narasayya, S. Chaudhuri and P. A. Bernstein, "Budget-aware Index Tuning with Reinforcement Learning," in *Proceedings of the 2022 International Conference on Management of Data*, New York, Association for Computing Machinery, 2022, p. 1528–1541.
13. X. Zhou, L. Liu, W. Li, L. Jin, S. Li, T. Wang and J. Feng, "AutoIndex: An Incremental Index Management System for Dynamic Workloads," in *2022 IEEE 38th International Conference on Data Engineering (ICDE)*, 2022, pp. 2196-2208.
14. S. Lai, X. Wu, S. Wang, Y. Peng and Z. Peng, "Learning an Index Advisor with Deep Reinforcement Learning," in *Web and Big Data*, Springer International Publishing, 2021, pp. 178-185.
15. R. M. Perera, B. Oetomo, B. I. P. 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)*, 2021, pp. 600-611.
16. R. M. Perera, B. Oetomo, B. I. P. Rubinstein and R. Borovica-Gajic, "HMAB: self-driving hierarchy of bandits for integrated physical database design tuning," *Proc. VLDB Endow.,* vol. 16, no. 2, p. 216–229, 2022.
17. P. Shor, "Algorithms for quantum computation: discrete logarithms and factoring," in *Proceedings 35th Annual Symposium on Foundations of Computer Science*, 1994, pp. 124-134.
18. L. K. Grover, "Quantum Mechanics Helps in Searching for a Needle in a Haystack," *Physical Review Letters,* vol. 79, no. 2, p. 325–328, 1997.
19. 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," *Proc. VLDB Endow.,* vol. 16, no. 9, pp. 2344-2353, 2023.
20. M. Matczak and T. Czochański, "Intelligent Index Tuning Using Reinforcement Learning," in *New Trends in Database and Information Systems*, Cham, Springer Nature Switzerland, 2023, pp. 523-534.
21. R. S. Sutton and A. G. Barto, "Reinforcement Learning: An Introduction," 2018, 2020. [Online]. Available: http://incompleteideas.net/book/RLbook2020.pdf.
22. S. Groppe, "Quantum Computing," [Online]. Available: https://www.ifis.uni-luebeck.de/~groppe/lectures/qc.
23. IBM, "Tutorials: Grover's algorithm," [Online]. Available: https://learning.quantum.ibm.com/tutorial/grovers-algorithm.
24. IBM, "IBM Quantum Learning: Grover's algorithm," [Online]. Available: https://learning.quantum.ibm.com/course/fundamentals-of-quantum-algorithms/grovers-algorithm.
25. Y. Wu, X. Zhou, Y. Zhang and G. Li, "Automatic Index Tuning: A Survey," *IEEE Transactions on Knowledge and Data Engineering,* pp. 1-20, 2024.
26. TPC, "Transaction performance council website," 1998. [Online]. Available: https://www.tpc.org/.

Formatted: Highlight

27. TPC, "TPC-H specifications," 2022. [Online]. Available: https://www.tpc.org/TPC_Documents_Current_Versions/pdf/TPC-H_v3.0.1.pdf.
28. Z. Sadri and L. Gruenwald, "A Divergent Index Advisor Using Deep Reinforcement Learning," in *Database and Expert Systems Applications*, Cham, Springer International Publishing, 2022, pp. 139-152.
29. I. Trummer and D. Venturelli, "Leveraging Quantum Computing for Database Index Selection," in *Proceedings of the 1st Workshop on Quantum Computing and Quantum-Inspired Technology for Data-Intensive Systems and Applications*, New York, Association for Computing Machinery, 2024, p. 14–26.
30. M. Kesarwani and J. R. Haritsa, "Index Advisors on Quantum Platforms," *Proc. VLDB Endow.,* vol. 17, no. 11, p. 3615–3628, 2024.
31. M. Matczak and T. Czochański, "Source Code: Intelligent Index Tuning Using Reinforcement Learning," [Online]. Available: https://github.com/Chotom/rl-db-indexing.
32. F. S. Melo and M. I. Ribeiro, "Q-Learning with Linear Function Approximation," in *Learning Theory*, Berlin, Heidelberg, Springer Berlin Heidelberg, 2007, pp. 308-322.
33. D. Dong, C. Chen, H. Li and T.-J. Tarn, "Quantum Reinforcement Learning," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics),* vol. 38, no. 5, pp. 1207-1220, 2008.
34. M. Ganger and W. Hu, "Quantum Multiple Q-Learning," *International Journal of Intelligence Science,* vol. 9, no. 1, pp. 1-22, 2019.
35. D. Barbosa, L. Gruenwald, L. d'Orazio and J. Bernardino, "Source Code: QRLIT," 2024. [Online]. Available: https://github.com/DBarbosaDev/QRLIT.