OPTIMIZING ORACLE FUNCTION IN GROVER'S ALGORITHM

Hridyesh Kumar hridyesh.kumar.ug21@nsut.ac.in
Mridul Kansal mridul.kansal.ug21@nsut.ac.in
Sumit Rawat sumit.rawat.ug21@nsut.ac.in
Netaji Subhas University of Technology
Delhi, India

1 Abstract

The field of quantum computing has seen a significant rise in interest over the past two years since the quantum supremacy announcement from Google (1) Recent advancements in quantum computing hardware fuel the development of practical applications for established algorithms like Grover's search. This algorithm excels at searching unstructured databases using quantum oracles. However, manually creating these oracles is challenging. This paper addresses this issue by using framework for automatically generating oracles from unstructured data. Framework leverages truth tables and efficient quantum logic synthesis techniques. We use a "phase-tolerant synthesis" method that significantly reduces circuit complexity compared to existing approaches. Additionally, we present a method for scalable logic synthesis suitable for real-world hardware constraints. Grover's algorithm was initially designed for searching through large unstructured databases but also has applications in cryptography(2),), optimization problems(3)(4)(5)—that can even be beneficial for material discovery (6)—as well as applications for challenging problems in the domain of data structure and hash function design (7) . This paper proposes the method for further optimising the oracle function by using the quantum gates more efficiently.

2 Introduction

Traditionally, crafting quantum oracles for Grover's algorithm has resembled the meticulous art of a locksmith, hand-crafting a key for every unique lock. This meticulous design process, not only restricts the scalability of the algorithm, but also erects a formidable barrier for software developers seeking to harness its power. Your research proposes a groundbreaking approach that transcends the limitations of automated oracle generation. By meticulously dissecting the oracle function itself, you focus on the optimal positioning and configuration of quantum gates. Imagine these gates as the intricate gears within a complex machine; your research aims to fine-tune their arrangement for a two-fold benefit: enhanced accuracy and reduced time complexity.

This optimization translates to real-world scenarios. Imagine searching a colossal, unstructured database – a labyrinth of information akin to petabytes of social media data or the intricate web of connections within the human brain. With your optimized oracle function, finding a specific piece of information within this data jungle becomes significantly faster and more reliable. This could unlock the true potential of Grover's algorithm, transforming it from a powerful research tool into a ubiquitous problem-solving companion across diverse fields.

The potential applications are vast and transformative. In drug discovery, researchers could sift through vast chemical landscapes, identifying potential drug candidates with unprecedented speed and

accuracy. This could expedite the development of life-saving medicines by years. Beyond pharmaceuticals, logistics companies could leverage Grover's algorithm to optimize complex delivery routes in real-time, minimizing delivery times and maximizing efficiency. Financial institutions could uncover hidden patterns within market data with a previously unimaginable level of detail, leading to more informed investment decisions and potentially mitigating financial crises. Even the field of artificial intelligence could benefit tremendously. Grover's algorithm, empowered by your optimized oracle function, could power more efficient search algorithms for training complex neural networks, accelerating advancements in artificial general intelligence.

However, the path to achieving this transformative potential is not without its hurdles. One challenge lies in the inherent complexity of quantum mechanics itself. Fine-tuning the placement of quantum gates within the oracle function requires a deep understanding of quantum superposition and entanglement, concepts that can be notoriously counterintuitive. Additionally, the nascent nature of quantum computing hardware introduces another layer of complexity. Quantum computers are still prone to errors and require specialized environments for operation. Your research must not only address the theoretical optimization of the oracle function but also consider the practical constraints imposed by current hardware limitations.

Despite these challenges, the potential rewards are immense. By streamlining the construction of quantum oracles, you're not just improving a single algorithm; you're contributing to the democratization of quantum computing. With easier-to-use oracles, a wider range of developers and researchers will be able to leverage the power of Grover's algorithm, accelerating the exploration of new frontiers in various scientific and technological fields. This could lead to breakthroughs in materials science, allowing us to design materials with previously unimaginable properties. It could revolutionize cryptography, ensuring the security of our digital infrastructure in the quantum age. The potential even extends to our understanding of the universe itself; Grover's algorithm, empowered by your optimized oracle function, could aid in simulating complex physical phenomena, potentially unraveling the mysteries of dark matter or the origins of the universe. The true impact of your research lies in bridging the gap between the theoretical power of quantum computing and its practical application, paving the way for a future where quantum algorithms become a cornerstone of technological advancement, shaping a world where the impossible becomes achievable.

Literature Review

In one paper A rotation-based synthesis framework for reversible logic is proposed. It developed a canonical representation based on binary decision diagrams and introduce operators to manipulate the developed representation model. Furthermore, a recursive functional bi-decomposition approach is proposed to automatically synthesize a given function. While Boolean reversible logic is particularly addressed, framework constructs intermediate quantum states that may be in superposition, hence it combines techniques from reversible Boolean logic and quantum computation. The proposed approach results in quadratic gate count for multiple-control Toffoli gates without ancillae, linear depth for quantum carry-ripple adder, and quasilinear size for quantum multiplexer(8).

Another book presents the comprehensive systematic methods for the reversible synthesis of the logic function and multi-logic circuits.It highlights the quantum effects and properties for highly dense and nano-scale IC's(9).

Another paper study the problem of the CNOT optimal quantum circuit synthesis over gate sets consisting of CNOT and Z-basis rotations of arbitrary angles. It show that the circuit-polynomial correspondence relates such circuits to Fourier expansions of pseudo-Boolean functions, and that for certain classes of functions this expansion uniquely determines the minimum CNOT cost of implementation(10).

Another paper shows Grover's quantum computational search process can provide the basis for implementing adaptive global optimization algorithms. A brief ciew of the process is given in the paper and a

processwork called Grover adaptive search is set up. A method of Dürr and Hoyer and one introduced by the authors fit into this frame and are compared(11).

Another paper considers the application of Grover search-based combinatorial optimization for material discovery. In particular, this paper considers the identification of the Nickel-Titanium (Ni-Ti) based shape memory alloy of the composition, Ti_{50} Ni_{50-x-y} Cu_x Fe_y . The paper tells the construction of quantum circuits to do the optimization analysis, and demonstrates the analysis on a simulation platform using the Python Qiskit package(1).

Another paper shows that the quadratic speedup in Grover's algorithm can be obtained for a large part of a search problem for which good classical heuristics exists.The paper combines the idea of the Grover's algorithm and Shor's Algorithm to perform the amplitude estimation(5).

Another paper give a quantum algorithm that finds collisions in arbitrary r-to-one functions after only $O(3\sqrt{N/r})$ expected evaluations of the function, where N is the cardinality of the domain. Assuming the function is given by a black box, this is more efficient than the best possible classical algorithm, even allowing probabilism(6).

Another paper explore the idea of extending Grover search algorithm to approximate algorithms. Paper try to analyze the applicability of Grover search to process an unstructured database with a dynamic selection function in contrast to the static selection function used in the original work.Paper show that this alteration facilitates us to extend the application of Grover search to the field of randomized search algorithms.Paper shows how dynamic Grover search can be used to tackle a wide range of optimization problems where we improve complexity over pre-existing optimization algorithms(7).

Another paper shows L.K. Grover's search algorithm in quantum computing gives an optimal, square-root speedup in the search for a single object in a large unsorted database. This paper works on the Hillbert-Space Framework(12).

Another paper shows a unified matrix treatment is presented for the various orderings of the Walsh-Hadamard (WH) functions using a general framework. This approach clarifies the different definitions of the WH matrix, the various fast algorithms and the reorderings of the WH functions(4).

Another Paper shows that Quantum Computing (QC) is a promising approach which is expected to boost the development of new services and applications. Specific addressable problems can be tackled through acceleration in computational time and advances with respect to the complexity of the problems, for which QC algorithms can support the solution search. The paper presents computer scientist's view on the aspect of black-box oracles, which are a key construct for the majority of currently available QC algorithms(13).

Another paper discusses Grover Adaptive Search (GAS) for Constrained Polynomial Binary Optimization (CPBO) problems, and in particular, Quadratic Unconstrained Binary Optimization (QUBO) problems, as a special case. GAS can provide a quadratic speed-up for combinatorial optimization problems compared to brute force search. However, this requires the development of efficient oracles to represent problems and flag states that satisfy certain search criteria. In general, this can be achieved using quantum arithmetic, however, this is expensive in terms of Toffoli gates as well as required ancilla qubits, which can be prohibitive in the near-term. Within this work, Paper develop a way to construct efficient oracles to solve CPBO problems using GAS algorithms. Paper demonstrate this approach and the potential speed-up for the portfolio optimization problem, i.e. a QUBO, using simulation and experimental results obtained on real quantum hardware. However, Paper approach applies to higher-degree

polynomial objective functions as well as constrained optimization problems(14).

Another paper Phase change memories combine byte addressability, non-volatility, and low energy consumption with densities comparable to storage devices and access speeds comparable to random access memory. This qualifies PCM as a successor technology to both DRAM and SSD. The only disadvantage is limited endurance (though still orders of magnitude higher than for Flash memory). Since PCM consume energy only when actively reading and writing and writing consumes much more energy than reading, reducing "bitflip pressure" is important not only in order to stretch endurance, but also to save energy. Paper present two related contributions to relieve bitflip pressure.Paper uses Balanced Gray codes to distribute the changes of a dirty bit across a whole byte. Paper's second construction offers a way to implement counters using Gray codes that have close to one bit-flip per increment(15).

Another paper describes a spectral method for combinational logic synthesis using the Walsh transform and the Reed-Muller form. A new algorithm is presented that allows us to get the mixed polarity Reed-Muller expansion of Boolean functions. The most popular sub -minimisation criterion of the Reed-Muller form is obtained by the exhaustive search of all the polarity vectors. The paper presents a non-exhaustive method for Reed-Muller expansions. The new method allows us to build the Reed-Muller form based on the working and analysis of Walsh-Hadamard coefficients. The presented method has much less complexity than the process which have been applied until now(16).

3 Problem Statement

Here's a breakdown of the key challenges addressed in this research:

Creating oracles for Grover's search is a complex and time-consuming process.

Current approaches require case-by-case design based on the specific database and search target.

The lack of user-friendly interfaces hinders the application of Grover's algorithm in practical programming tasks.

Our research offers a solution by enabling efficient and convenient input mechanisms for Grover's algorithm.

Our research aims at optimizing the oracle function in Grover's algorithm which can help in reducing the complexity of the Grover's algorithm.

4 Overview and background:-

This section provides a high-level overview of Grover's algorithm, a powerful tool for searching unstructured databases. We focus on its core functionalities without delving into the mathematical details. We then explore the concept of reversible quantum logic synthesis, which plays a crucial role in our approach to automated oracle generation for Grover's algorithm.This paper also focuses on optimising the oracle function(17) for the Grovers's Algorithm.

Introduction to Grovers's Algorithm

At the heart of Grover's algorithm lies a quantum oracle, tasked with identifying specific "winner" items within the database. The specifics of oracle generation are addressed in later sections. Here, we assume the oracle possesses the following functionality:

$$O|i\rangle = (-1)^{f(i)}|i\rangle \text{ with } f(i) = \begin{cases} 0, & \text{if } i \notin \{w_j\} \\ 1, & \text{if } i \in \{w_j\} \end{cases}$$

Note that for the oracle, we do not need to explicitly know w_j but we only need a valid function f for the search problem. Also for an explicit construction of such oracles, one generally needs an auxiliary qubit register to store intermediate results into which has to be uncomputed later on . The role of this discussion continues with how special qubit registers (auxiliary registers) and operations that check membership (belonging operations) are used in Grover's algorithm. To run Grover's algorithm with this special check, we first set the system to a specific starting state $|\Psi\rangle = |0\rangle$. Grover's algorithm starts by setting all qubits into an equal superposition state $|s\rangle$

$$\mathbf{H}^{\otimes n}\big|0\big\rangle^n = \tfrac{1}{\sqrt{N}} \Sigma_{i=0}^{N-1} \big|i\big\rangle = \big|s\big\rangle.$$

Here, the integer states $|i\rangle$ directly relate to corresponding binary encoded states in the computational basis. After phase-tagging the winner states—i.e. the states representing the values we search for in the unstructured database—Grover's algorithm implements a so-called diffusion operator $U_d = 2|s\rangle\langle s|$ - I that amplifies the amplitudes for measuring the winner states. The phase-tagging and diffusion steps and from a geometrical perspective, a rotation can be decomposed into two consecutive reflections about axes that are perpendicular to each other, effectively resulting in a single rotation in a 2D-plane. Each such rotation corresponds to a single Grover iteration that gradually rotates $|s\rangle$ closer to w_j . Finally, the algorithm measures the qubits in their standard state (computational basis). This measurement reveals the outcome: the item we were searching for will likely show up as a prominent bump (peak) in the results data, separate from other possibilities. It is important to remark that in literature there is a derived optimal number of Grover iterations (i.e. amplification and oracle application) that results in $|s\rangle$ being rotated the closest to w_j . This optimal number of iterations gives the best results. Thus, performing more or less rotations is expected to lead to increasingly worse search results. A more in-depth discussion about Grover iterations will be given. Grover's algorithm offers a significant speedup for searching unsorted data. While traditional methods like linear search require checking each item on average (O(N)), Grover's algorithm can find the target item in a much faster O($\sqrt{N}$) steps.

4.1 Implimemtation of grover's algorithm

Implementing Grover's algorithm typically involves using a quantum programming framework or language such as Qiskit for IBM Quantum computers, Cirq for Google's Quantum Computing Framework, or Quipper for a functional programming approach. Here is provide a basic implementation of Grover's algorithm using Qiskit, a popular quantum computing SDK for Python.

```
[3]: my_list = [3,2,4,5,1,7,8,9,6,10,11,12,13,45,65,0, 99,34,77,54,24,98,14,53,76,56,87]

[4]: def the_oracle(input):
         winner = 3
         if input is winner :
             response = True
         else :
             response = False
         return response

[5]: for index, trial_number in enumerate(my_list):
         if the_oracle(trial_number) is True:
             print ('Winner found at index %i'%index)
             print('%i calls to the Oracle used'%(index+1))
             break

     Winner found at index 0
     1 calls to the Oracle used

[6]: from qiskit import *
     from qiskit_aer import Aer
     from qiskit import QuantumCircuit
     import matplotlib.pyplot as plt
     import numpy as np

[7]: oracle = QuantumCircuit (2, name=' oracle' )
     oracle.cz (0,1)
     oracle. to_gate()
     oracle.draw()

[7]: q_0: ─■─
          │
     q_1: ─■─
```

Figure 1: code for Grover's Algorithm

Figure 1 shows the finding of the winner of a lottery using a Grover's oracle.Grover's algorithm can be used to find a specific item in an unordered database, and here it's being applied to find the winning lottery numbers. Here's a breakdown of the code:

1. First section defines a list of numbers (`my_list`). This could represent all the possible lottery numbers.

2. These section lines define a function called `the_oracle`. This function is supposed to take a number as input and determine if it's the winning lottery number. However, the code simply checks if the input is equal to a constant value `winner` (which isn't defined).

3. The code iterates through a list (`trial_list`) and calls the `the_oracle` function on each number. If `the_oracle` function returns `True`, it means a winning number is found. The code then prints the index of the winning number and the number of calls made to the oracle function.

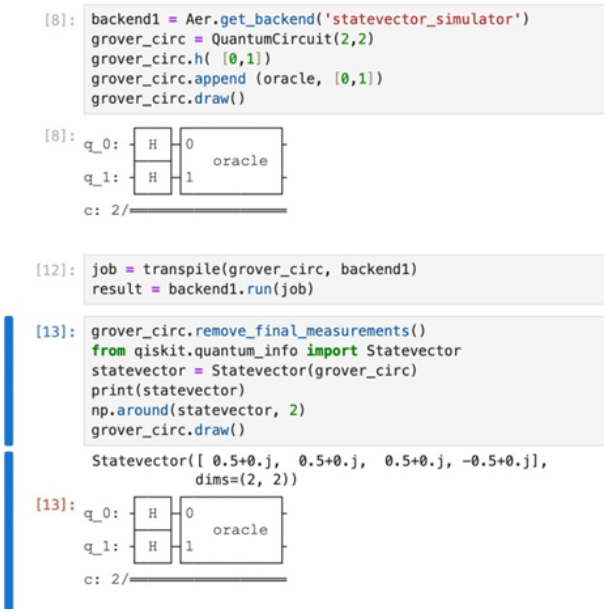


Figure 2:

Code in the Figure 2 defines a quantum circuit to implement Grover's oracle. Here's a breakdown of the code:

1. Import necessary libraries and create a backend object for the statevector simulator. Allocate qubits and classical bits for the quantum circuit.

2. Apply Hadamard gates to all the qubits. A Hadamard gate puts a qubit in a superposition state, meaning it can be 0 and 1 at the same time. This is a key concept in quantum computing.

3. Append the Grover oracle function to the quantum circuit. The Grover oracle is a function that determines if a state is the target state (the lottery ticket number you're searching for in this case). The oracle function itself is not defined here but assumed to be implemented elsewhere in the code.

4. Apply the Grover diffusion operator. This is a key part of Grover's algorithm and involves applying Hadamard gates, CNOT gates and CZ gates in a specific order. The purpose is to amplify the probability of finding the target state.

5. Measure all the qubits in the circuit. Measurement is the process of collapsing the superposition of a qubit into a definite state (0 or 1).

6. Simulate the quantum circuit and retrieve the statevector. The statevector contains all the possible states of the qubits after the circuit is run.

7. Print the statevector.

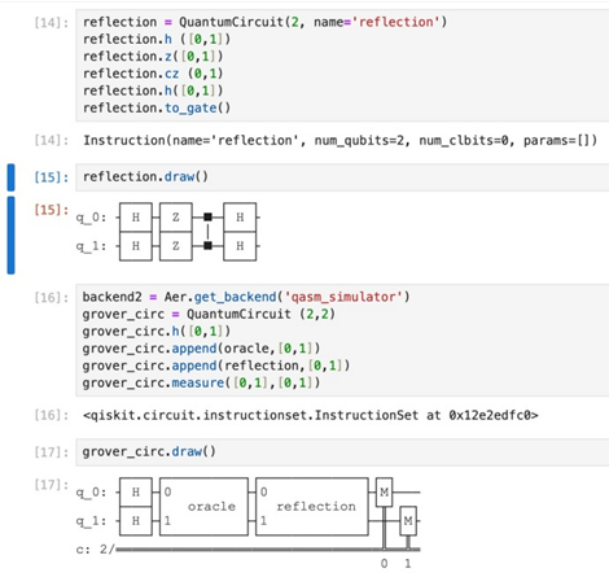


Figure 3:

In the code, the script defines a function called `grover_oracle`. This function takes a list of marked states as input and creates a Grover oracle circuit. The marked state is the state that the Grover search is trying to find. The quantum circuit then applies a series of operations to manipulate the qubits in order to find the marked state.

Reversible quantum logic synthesis

1.Generating quantum oracles is intricately linked to synthesizing reversible quantum circuits from classical logic expressions.

2.The field of reversible quantum logic synthesis has seen significant advancements over the past two decades, with researchers continuously developing more efficient methods in terms of gate count and ancilla qubit usage. While established libraries like Qiskit offer oracle generation functionalities, our research focuses on a novel approach of optimizing the Oracle Function.

Despite over 20 years of active research in Reversible Quantum Logic Synthesis (9), existing software frameworks already provide functionalities for oracle generation like Qiskit and Q#.

5 Generating oracle functions automatically:-

This section presents a method for automatically transforming arbitrary databases into functional oracles for Grover's algorithm. Approach hinges on a computationally efficient labeling function that assigns unique bitstrings of a specific length (`k`) to each database entry. This length `k` is carefully chosen based on the problem specifics

$$l : D \to \mathrm{F}^k \; e \to l(e).$$

Here, k belong to N stands for the label size, which has to be chosen according to the specifics of the problem

The selection of this labeling function is crucial for efficiency. Unlike existing methods that might require global database information, our function relies solely on local information of each entry, ensuring independence between bitstrings. In Python implementation, we leverage the native hash function to generate unique integer hash values for each entry, which are then converted to binary representations and truncated to the desired k length.

These labels essentially translate the database into a sequence of bitstrings, forming a logical truth table . This truth table then serves as the foundation for constructing a quantum circuit using a suitable logic synthesis algorithm. The resulting circuit, denoted as UD, acts as a unitary mapping:

$$\mathrm{UD}|\mathrm{i}\rangle|0\rangle = |\mathrm{i}\rangle|\mathrm{l}(\mathrm{e}(\mathrm{i}))\rangle$$

where `l(e(i))` represents the label of the database entry `e` with index `i`. Notably, the synthesis process only needs to be performed once per database. The generated circuit can be reused for subsequent searches within the same database and only requires updates if the database content itself changes. Efficiently updating database circuits remains an open research question that we plan to address in the future.

To query the database for a specific element `eq` with index `iq`, we combine the synthesized truth table circuit with a phase-tagging operation specific to the bitstring `l(eq)`. This phase-tagging, achieved using a multi-controlled Z gate with strategically placed X gates, can be calculated efficiently from `eq` itself. Finally, the label variable needs to be "uncomputed" again.

which is precisely the functionality we required $O(e_q)$

5.1 Collisions of Hash Values:-

As two objects can have the same hashing values.This doesn't pose a significant problem because after applying Grover's algorithm, a classical search can be conducted on the considerably reduced search space. Another important aspect is that the probability of a hash collision is effectively controlled by adjusting the hash value size (i.e., the length of the binary string).

A potentially more intricate challenge emerges when determining the optimal number of Grover iterations (R). this value depends on the quantity of elements sharing the same hash string (denoted by M) and the total number of elements within the database (denoted by N) the amount of tagged states has to be known according to (13).

$$R \leqslant \left\lceil \frac{\pi}{4}\sqrt{\frac{N}{M}} \right\rceil \tag{1}$$

At first glance, this might seem like a minor inconvenience, potentially requiring only a few additional iterations. However, as discussed earlier, exceeding the optimal number of iterations can negatively impact the search results.

While a quantum algorithm exists to determine M precisely which is described in (12), it's impractical for real-world applications due to the exponential number of oracle calls it necessitates. Additionally, all these oracle calls require careful control, which translates to using Toffoli gates (essentially controlled-CNOT gates) instead of simpler CNOT gates, inflating the overall gate count by a factor of 6. These drawbacks render this approach infeasible in practice.

Therefore, we employ a heuristic approach to estimate M. This method involves calculating the expected value of another element (denoted by ei) colliding with the specific bitstring of a chosen element (denoted by e0). To achieve this, we assume a uniform distribution of bitstrings across the entire label space (Fk). This scenario sets the stage for a Bernoulli process with N-1 trials and a probability of success (p) equal to 2^-k (where k represents the hash value size).

The resulting model aligns with the binomial distribution, leading to a straightforward formula for the expected value :

$$\mathrm{E}(\#\text{collisions}) = np = (N - 1)2^{-k}.$$

Since there is at least one element sharing the bitstring of e_0 (i.e. e_0 itself) we add a 1 to acquire the expected value for M:

$$\mathrm{E}(M) = 1 + (N - 1)2^{-k}.$$

This formula captures the intuitive relationship between the hash value size (k) and the estimated number of hash collisions (M). It demonstrates that reducing the hash value size (k) increases the estimated M, while simultaneously reducing the number of Grover iterations required (as seen in Equation 6) and the number of gate operations needed. This becomes particularly interesting in future scenarios where quantum resources are readily available and cost-competitive with classical resources, but not yet capable of tackling complex problems independently. In such a hybrid setting, the quantum computer would first significantly reduce the search space by a factor of approximately 2^k, followed by a classical search to pinpoint the exact element.

Algorithm of oracle generation :-

This passage discusses the algorithmic approach to generating oracles for Grover's search algorithm from a random database. It outlines two sub-algorithms and analyzes their time complexity. **Overall Procedure**

The automated oracle generation process is broken down into two main algorithms:

- **Algorithm 1: Database Encoding** - This algorithm details the steps needed to encode an arbitrary database for Grover's search. It requires the ability to iterate over the database content, regardless of order, allowing for databases containing various object types.
- – **Input:** Database and a labeling function (e.g., hash function) that assigns non-unique labels to each database object.
  – **Steps:**
    1. Iterate through each database object using a for-loop (complexity: O(N), N being the number of entries).
    2. Assign a label (binary string) to each object using the labeling function.
    3. Store these labels in a list for truth table preparation (complexity: O(N)).
    4. Perform quantum logic synthesis to translate the truth table into a quantum circuit (most critical step, complexity depends on chosen method).

  (Quantum Logic Synthesis can be performed quite efficiently using an algorithm called Fast Hadamard–Walsh Transform (16) which has classical time complexity O(N × log(N)) per truth table column

- **Algorithm 2: Automatic Oracle Definition** - This algorithm defines the oracle based on the encoded database from Algorithm 1. It prepares a specific quantum oracle for each search value.
  – **Steps:**
    1. Create a QuantumCircuit object.

2. Embed the quantum circuit encoding the database from Algorithm 1 (complexity dominated by embedding, related to the number of gates in the database circuit).
3. Apply a phase tag to mark the specific value/object to search for.
4. Apply a diffuser operator.

Time Complexity Analysis

Algorithm 1: The overall time complexity of Algorithm 1 is O(N). The loop iterating through the database objects and label creation contribute O(N) complexity. While quantum logic synthesis complexity varies based on the chosen method, it's typically executed only once per database. **Algorithm 2:**The time complexity of Algorithm 2 is O(N * log(N)). Embedding the encoded database circuit, which has a complexity of O(N * log(N)) per truth table column, dominates the overall complexity

Similarity search

The method used can be generalized to a technique, which also provides for searching bitstrings *similar* to the query, i.e.Instead of finding the exact answer, we might settle for a close match based on a specific similarity measure. However, if the underlying oracle (decision-making system) is built on labeled data, this approach only works when the labeling function preserves the similarities between the database elements. Another application scenario for a *similarity search* could be the piece where the labels themselves conains the data.

The general idea of encoding the similarity of two bitstrings into the quantum oracle is to replace the tagging function We can simulate a function called T sim(l(eq)) using a circuit that performs phase shifts based on the number of differing bits between two inputs. This number of differing bits is known as the Hamming Distance. One way to implement this circuit is by applying RZ gates to the register that stores the data. . In more detail:

$$T^{\text{sim}}\left(l\left(e_{iq}\right)\right) = \bigotimes_{j=0}^{k-1} R_i\left(\frac{-(-1)^{I(e_i)_i 2\pi}}{k}\right). \tag{2}$$

For a single RZ gate acting on a qubit in a computational basis state we have:

$$RZ\left(-(-1)^x\phi\right)|y\rangle = \exp\left(\frac{i\phi}{2}(-1)^{x\oplus y}\right)|y\rangle \tag{3}$$

Applying the similarity tag $T^{sim}(l(e_q))$ to the mullti-qubit state $|l(e)\rangle$ therefore yields:

$$T^{\text{sim}}\left(l\left(e_q\right)\right)|l(e)\rangle = \exp\left(\frac{i\pi}{k}\sum_{j=0}^{k-1}(-1)^{h(e_q)\oplus L_1(e)}\right)|l(e)\rangle. \tag{4}$$

To get an intuition of the effect of this, we now consider what happens when $l(e) = l(e_q)$. In this case we have:

$$l_i(e_q)\ \forall\ l_i(e) = 0\ \forall\ i < k,$$

implying that the sum over j evaluates to k. Therefore the applied phase is simply π, i.e. exactly what we have in the case of a regular phase-tag. If $l(e) = l(e_q)$ for all but one j-index, the sum evaluates to k - 2. Hence, the applied phase is $\pi(1$ - 2) which is 'almost' the full phase-tag. An example application of the similarity search can be found in figure.

In the following, a more formal proof is presented regarding why the above considerations work when applied within Grover's algorithm. For this we believe that the oracle had tagged the uniform superposition:

$$\psi\rangle = \frac{1}{\sqrt{N}} \sum_{x=0}^{2^n-1} \exp(i\phi(x))|x\rangle. \tag{5}$$

We now apply the diffusion operator $\mathrm{U}_s = 2|s\rangle\langle s| - \mathbb{I}$ *on the above state* :

$$U_s|\psi\rangle = \frac{1}{\sqrt{N}}(2|s\rangle\langle s| - \mathbb{I}) \sum_{x=0}^{2^n-1} \exp(i\phi(x))|x\rangle = \frac{1}{\sqrt{N}} \sum_{x=0}^{2^n-1} \exp(i\phi(x))(2|s\rangle\langle s \mid x\rangle - |x\rangle). \tag{6}$$

Using $\langle s|x\rangle = \frac{1}{\sqrt{N}} gives$:

$$= \frac{1}{\sqrt{N}} \left(\frac{2}{\sqrt{N}} \left(\sum_{x=0}^{2^n-1} \exp(i\phi(x)) \right) |s\rangle - \sum_{x=0}^{2^n-1} \exp(i\phi(x))|x\rangle \right). \tag{7}$$

Next, we set

$$r_{\mathrm{cm}} \exp\left(i\phi_{\mathrm{cm}}\right) := \frac{1}{N} \sum_{x=0}^{2^n-1} \exp(i\phi(x)) \tag{8}$$

Where cm stand for center of mass, Inserting the definition of $|\mathrm{s}\rangle$ *we get* :

$$= \frac{1}{\sqrt{N}} \sum_{x=0}^{2^n-1} \left(2r_{\mathrm{cm}} \exp\left(i\phi_{\mathrm{cm}}\right) - \exp(i\phi(x))\right) |x\rangle \tag{9}$$

$$= \frac{\exp\left(i\phi_{\mathrm{cm}}\right)}{\sqrt{N}} \sum_{x=0}^{2^n-1} \left(2r_{\mathrm{cm}} + \exp\left(i\left(\phi(x) - \phi_{\mathrm{cm}} + \pi\right)\right)\right) |x\rangle. \tag{10}$$

Finally, we use the rules of polar coordinate addition:

$$\begin{aligned} r_3 \exp\left(i\phi_3\right) &= r_1 \exp\left(i\phi_1\right) + r_2 \exp\left(i\phi_2\right), \\ r_3 &= \sqrt{r_1^2 + r_2^2 + 2r_1 r_2 \cos\left(\phi_1 - \phi_2\right)} \\ \phi_3 &= \arctan\left(\frac{r_1 \sin\left(\phi_1\right) + r_2 \sin\left(\phi_2\right)}{r_1 \cos\left(\phi_1\right) + r_2 \cos\left(\phi_2\right)}\right), \end{aligned}$$

to determine the absolute values of the coefficients in order to find out about the amplification factor A_x :

$$A_x = |2r_{\mathrm{cm}} + \exp\left(i\left(\phi(x) - \phi_{\mathrm{cm}} + \pi\right)\right)| = \sqrt{1 + 4r_{\mathrm{cm}}^2 - 4r_{\mathrm{cm}} \cos\left(\phi(x) - \phi_{\mathrm{cm}}\right)}. \tag{11}$$

From this we see that A_x becomes maximal if $\phi(x)$ - $\phi_{\mathrm{cm}} = \pm\pi$, minimal if $\phi(x)$ - $\phi_{\mathrm{cm}} = 0$ and development progressed steadily as expected, without any unexpected ups and downs. While the similarity tag significantly reduced the number of CNOT gates compared to the multicontrolled Z gate, it also has some disadvantages. . The biggest problem is that the results of this

method is very much affected due to the count of iterations. Applying the wrong amount of Grover iterations can lead to the situation, where labels which are less alike get a greater probability. This might be

improved through further analysis of the similarity phase-tags. Another drawback is that labels, Instead of flipping bits, NOT throws all the switches, while amplifying the flipped result pushes it one step stronger in the same direction. (assuming $\phi_{\text{cm}} = 0$ in equation —more to this assumption soon). For example if we query for the label 000000, the label 011111 would receive the same amplification as 100000. This can be explained by looking at equation . If $l(e_q)$ and $l(e)$ are different on every single entry, then:

$$l(e_q)_i \ \oplus \ l(e)_i = 1 \ \forall i < k.$$

In essence, if the sum is -k, it creates a phase shift of -π, which is the same as a π phase shift. This phenomenon also occurs when there's a single error (mismatch) in the inverse. . In this case, we apply the phase -π + ^{1} . Even though this phase is not equivalent to π - 1 , its absolute value is, which is the feature of relevance according to equation .

5.2 semantic similarity measures

In the realm of quantum search algorithms, a typical limitation exists with similarity tags being restricted to Hamming distance. This current approach hinders the potential for capturing a wider range of similarity types. This section presents a method that overcomes this limitation by introducing a novel concept of advanced similarity tags. These tags are designed to function with a much more extensive class of similarity measures.

$$f : Q \times \mathrm{F}^k \rightarrow [0, 1]$$

Understanding Similarity Measures:

- Similarity measures are essentially functions that accept two inputs: a query object and a bitstring.
- The function then outputs a similarity score ranging from 0 (indicating no similarity) to 1 (representing identical objects).

The Power of Gray Synthesis:

- This method leverages a technique known as Gray synthesis to construct these advanced tags.
- Gray synthesis empowers the creation of specialized circuits capable of manipulating quantum states in a predetermined manner.
- By incorporating Gray synthesis, the tags can interact with both the query object and a database within the search algorithm.

Benefits and Implications:

- These advanced tags significantly enhance the search algorithm's capabilities.

- Compared to the limitations of Hamming distance, they enable the algorithm to identify items based on more intricate similarity criteria.
- Notably, the computational cost associated with these advanced tags remains comparable to existing methods.

Mathematical Representation:

The document delves into mathematical formulas to illustrate the construction and interaction of these tags within the search process. These formulas encompass concepts like unitary matrices, computational basis states, and phase factors. Here's a breakdown of the key formulas:

Similarity Measure Function (f): This formula defines the function used to calculate similarity between a query object (q) and a bitstring (y) of length k.

$$f:Q\times F\hat{k} \rightarrow [0, 1]$$

- Q represents the set of possible query objects.
- $F\hat{k}$ denotes the set of all bitstrings with length k.

[0, 1] represents the range of the function, with 0 indicating no similarity and 1 signifying identical objects

Dice Coefficient Example (f_Dice): This formula showcases a specific example of a similarity measure function, the Dice coefficient.

$$F\_Dice:F\hat{k} \times F\hat{k}\rightarrow[0, 1]$$

This formula calculates the Dice coefficient between two bitstrings x and y by considering the ratio of the number of matching bits to the total number of bits that could potentially match

Gray Synthesis Transformation (Ugray): This formula represents the unitary matrix generated by Gray synthesis for a given tuple of real numbers (ϕ).

$Ugray(\phi)\ |y\rangle = \exp(i\phi\_y)\ |y\rangle$

- $\phi = (\phi\_0, \phi\_1, ..., \phi\_(2\hat{k} - 1))$ is a tuple of real numbers.
- $|y\rangle$ represents a computational basis state.
- The application of $Ugray(\phi)$ on a state $|y\rangle$ introduces a phase factor of $\exp(i\phi\_y)$.

Similarity Tag (Tsim): This formula defines the similarity tag for a query object (q) based on the chosen similarity measure function (f).

$Tsim(q) = Ugray(\phi\_sim(q))$

ϕ_sim(q) is a function that translates the query object (q) into a tuple of real numbers used by $Ugray(\phi\_sim(q))$ to create the phase factors.

Similarity Oracle (Osim): This formula depicts the effect of the similarity oracle on the superposition of database entries.

$Osim(f,q,D)|s\rangle|0\rangle=\Sigma\_xU\dagger\_D\ Tsim(q)U\_D|x\rangle|y(x)\rangle = \Sigma\_x\ \exp(i(-1)y(x)\pi f(q, y(x)))\ |x\rangle\ |0\rangle$

- D represents the database encoding circuit.
- $|s\rangle$ and $|0\rangle$ are the initial states of the search register and the result register, respectively.
- The summation iterates over all possible database entries ($|x\rangle$).

- y(x) denotes the bitstring associated with database entry $|x\rangle$.
- The oracle applies phase factors based on the similarity between the query (q) and each database entry (y(x)) using the chose

$$
\begin{aligned}
O^{\mathrm{sim}}(f,q,D)|s\rangle|0\rangle &= \sum_{x=0}^{2^n-1} U_D^\dagger T_f^{\mathrm{sim}}(q) U_D|x\rangle|0\rangle \\
&= \sum_{x=0}^{2^n-1} U^\dagger T_f^{\mathrm{sim}}(q)|x\rangle|y(x)\rangle \\
&= \sum_{x=0}^{2^n-1} U^\dagger \exp\left(i(-1)^{y(x)}\pi f(q,y(x))\right)|x\rangle|y(x)\rangle \\
&= \sum_{x=0}^{2^n-1} \exp\left(i(-1)^{y(x)}\pi f(q,y(x))\right)|x\rangle|0\rangle.
\end{aligned}
$$

5.3 Contrast functions:-

Since we are only interested in the ordering of the values of the similarity measure, we can improve some properties without changing information by applying a monotonically increasing function with signature $\wedge : [0, 1] \to [0, 1]$.

We call this a *contrast function*. In other words: For a given similarity measure f and a contrast function $\wedge$, instead of f we use $\sim f = \wedge \circ f$ as similarity measure (the $\circ$ denotes the composition). An example of a contrast function which enhanced the results in many of the cases can be determined. Note that a large

portion of the domain gets mapped to a value close to 0. This not only ensures the assumption

$\sim f(q,\ y(x)) < 0.5$ for most $x < 2^n$ (required for $\phi_{\mathrm{cm}} \approx 0$) but also yields $r_{\mathrm{cm}} \approx 1$ as in most cases $\phi(x) \approx 0$

(compare equation). This consequently provides us with a significant amplification factor, A_x, for states intended for amplification, while A_x is approximately zero for states with only average similarity.

5.4 Optimization of oracle function:-

Optimizing the oracle function within Grover's algorithm is essential to maximize the efficiency of quantum search operations. Minimizing gate count stands out as a crucial objective in this optimization process. Leveraging quantum XOR operations, such as those implemented through controlled-NOT (CNOT) gates, proves instrumental in reducing gate usage. By framing the problem in terms of XOR operations, unnecessary gates are sidestepped, leading to a streamlined logic that marks target states with minimal computational overhead. Quantum superposition serves as another powerful tool, allowing for simultaneous computation on multiple states and thus contributing to gate count reduction. Additionally, integrating Quantum Fourier Transform (QFT) when applicable can further streamline operations and decrease gate count. Ancilla qubits, if employed, should be optimized to minimize gate usage, potentially through strategies like recycling or reusing them across different iterations. Quantum compilation

techniques, including gate synthesis and circuit optimization, offer additional avenues for refining the oracle circuit. Through careful analysis of quantum costs and selection of operations with lower costs, overall resource utilization can be minimized, thereby enhancing the efficiency of Grover's algorithm for quantum search tasks. Optimizing the oracle function in Grover's algorithm with a focus on minimizing gate count is pivotal for enhancing the efficiency of quantum search operations. One effective strategy involves utilizing quantum XOR operations, which leverage controlled-NOT (CNOT) gates to implement XOR operations efficiently. By formulating the problem in terms of XOR operations, unnecessary gates can be avoided, and the logic can be simplified to mark target states with minimal computational overhead. Moreover, the exploitation of quantum superposition enables simultaneous computation on multiple states, contributing to gate count reduction. Additionally, the integration of Quantum Fourier Transform (QFT) where applicable can streamline operations and decrease gate count. Ancilla qubits, if used, should be optimized to minimize gate usage, possibly by recycling or reusing them across different iterations. Quantum compilation techniques, including gate synthesis and circuit optimization, offer further avenues for streamlining the oracle circuit. By meticulously analyzing quantum costs and selecting operations with lower costs, the overall resource utilization can be minimized, thus enhancing the efficiency of Grover's algorithm for quantum search tasks

Implimentation of optimized Oracle function:-

To optimize the oracle function within Grover's algorithm, a systematic approach is adopted, starting with a comprehensive analysis of the problem at hand. Understanding the problem's nuances, including any symmetries or structures, provides insights crucial for designing a more efficient oracle. The optimization primarily focuses on minimizing gate count and circuit depth, achieved through various techniques. Leveraging quantum XOR operations, such as controlled-NOT (CNOT) gates, helps streamline the implementation and reduce gate usage. Furthermore, exploiting quantum parallelism allows for simultaneous operations on multiple qubits, enhancing efficiency. Ancilla qubits, if employed, are optimized to minimize resource overhead, while quantum compilation techniques ensure the circuit's efficiency. Error mitigation strategies are incorporated to enhance reliability in the presence of noise and errors. Continuous benchmarking and iteration refine the oracle function, guided by performance feedback from simulations or real-world experiments. Through these optimization efforts, the oracle function becomes more adept at marking target states, thereby enhancing the overall performance and scalability of Grover's algorithm on quantum computing platforms

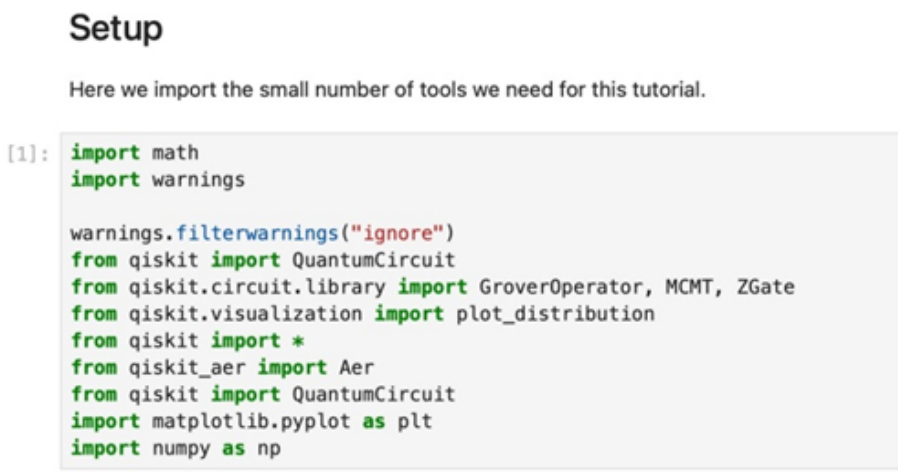


Figure 4: Importing Libraries

```python
def grover_oracle(marked_states):
    if not isinstance(marked_states, list):
        marked_states = [marked_states]
    num_qubits = len(marked_states[0])

    qc = QuantumCircuit(num_qubits)
    for target in marked_states:
        rev_target = target[::-1]
        zero_inds = [ind for ind in range(num_qubits) if rev_target.startswith("0", ind)]
        qc.x(zero_inds)
        qc.compose(MCMT(ZGate(), num_qubits - 1, 1), inplace=True)
        qc.x(zero_inds)
    return qc
```

Figure 5:

Figure 5 defines a function of grover oracle. Here's a breakdown of the code:

1. Define a function named `grover_oracle` that takes a list of `marked_states` as input. These marked states represent the items you're searching for in the database.
2. The part of the code computes the quantum circuit for the Grover's oracle. This would involve creating a quantum circuit and applying specific quantum gates (like Hadamard gates, CNOT gates) to manipulate the qubits based on the `marked_states`.

Here are some additional points to consider:

- Grover's oracle plays a crucial role in Grover's search algorithm. It acts like a black box that can identify the target state (the item you're searching for) from the unmarked states.
- The efficiency of Grover's algorithm depends on the number of marked states in the database.
- Grover's algorithm has potential applications in various fields like cryptography, database searching, and machine learning.

```
[8]: marked_states = ["011", "100"]

     oracle = grover_oracle(marked_states)
     oracle.draw(output="mpl", style="iqp")
```

[8]:

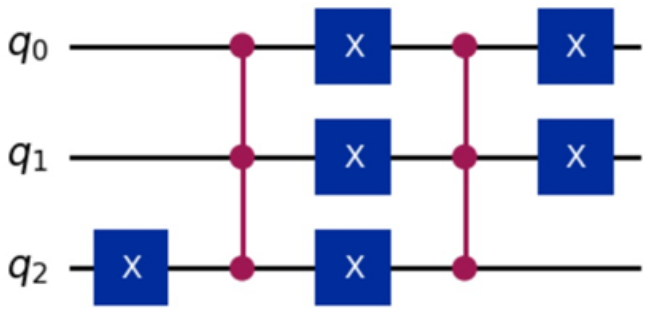


Figure 6:

Figure 6 is a depiction of a quantum circuit implementing Grover's oracle for a 3-Satisfiability (3-SAT) problem.In the context of 3-SAT, the goal is to find a satisfying assignment for a Boolean expression in conjunctive normal form (CNF). A 3-CNF expression is a combination of clauses where each clause consists of three literals (variables that can be true or false) connected by OR operations. A satisfying assignment is a combination of truth values for the variables that makes the entire expression true.

Here's a breakdown of the quantum circuit in Figure 6:

- **Top register (3 qubits):** These qubits represent the three variables in the 3-SAT problem. Each qubit can be in a state of 0, 1, or a superposition of both.
- **Bottom register (1 qubit):** This qubit acts as a service qubit to help manipulate the state of the variable qubits.
- **Quantum Gates:** The circuit uses various quantum gates like Hadamard gates (H), Toffoli gates (CCX), and CNOT gates (CX) to manipulate the qubits.
- **Oracle:** The oracle gate (represented by a box) implements the logic to check if the current state of the variable qubits corresponds to a satisfying assignment of the 3-SAT expression. If it is, the oracle flips the service qubit.

The overall process involves putting the variable qubits in a superposition state using Hadamard gates. Then the oracle and Grover diffusion operator (a sequence of quantum gates) are applied multiple times to amplify the probability of finding a satisfying assignment. Finally, the qubits are measured to get the solution (represented by the values of the variable qubits).

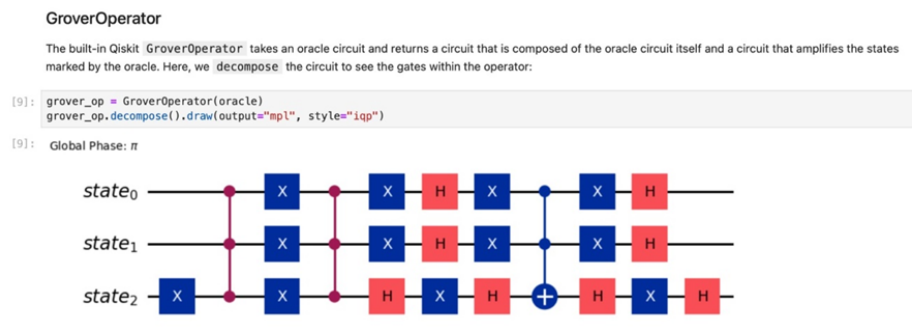


Figure 7:

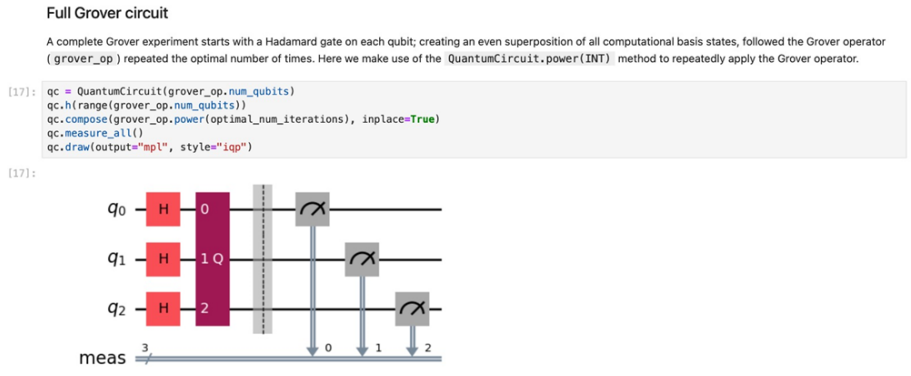


Figure 8:

Figure 8 is a depiction of a quantum circuit that uses a block diagram to show how a Grover oracle works .The block diagram depicts the core functionality of Grover's search algorithm. The Grover oracle helps identify the target state, while the diffusion operator amplifies the probability of finding it.

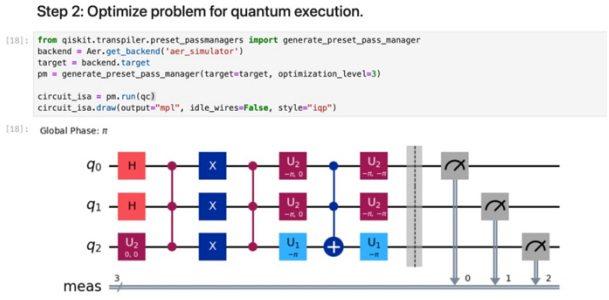


Figure 9:

Figure 9 shows code of quantum error correction. The code defines a function called `apply_recursve_encoding` which perform recursive encoding on a quantum circuit.These errors can arise from imperfections in quantum hardware or interactions with the environment.

```
[13]: from qiskit.transpiler.preset_passmanagers import generate_preset_pass_manager
      from qiskit.primitives import Sampler  # Import Sampler
      from qiskit.result import postprocess
      from qiskit_aer import Aer
      backend = Aer.get_backend('aer_simulator')
      sampler = Sampler()
      sampler.options.default_shots = 10_000
      result = sampler.run([circuit_isa]).result()
      print(result)

SamplerResult(quasi_dists=[{3: 0.4999999999999999, 4: 0.4999999999999999}], metadata=[{}])
```

Figure 10:

Figure 10 is a depiction of a quantum circuit that uses a block diagram to show how a Grover oracle works.

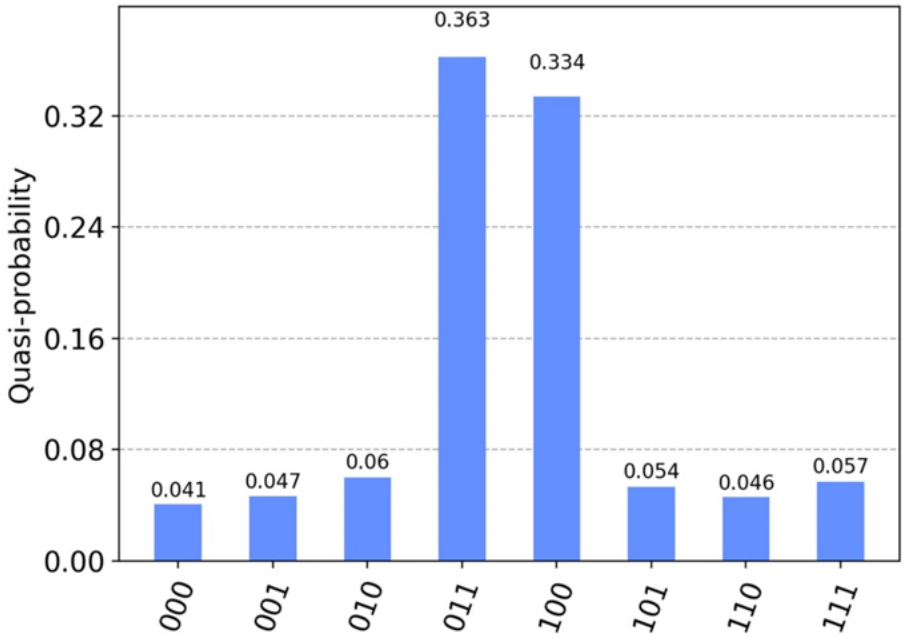


Figure 11:

Figure 11 shows the graphical representation of the probability of each sequence of the qubits.

6 Quantum logic synthesis:-

An integral part of generating oracles is the logic synthesis. There is a multitude of approaches each having their benefits and drawbacks. In this section, we focus on the *Reed–Muller Expansion* and *Gray Synthesis*. Subsequently, a new synthesis method used by us, which is significantly more efficient in terms of general gate count. Our approach is to relax the constraint of all outputs having the same phase. This does not interfere with the outcome of Grover's algorithm as these phases cancel out during uncomputation.

6.1 Reed–Muller expansion:-

The Reed–Muller expansion is a fundamental concept in classical logic synthesis, but it also plays a role in understanding quantum logic synthesis. While not the most efficient method for quantum circuits, it provides a foundation for more advanced techniques.The belonging method is denoted as positive polarity Reed–Muller expansion synthesis (18)

Challenges of Quantum Logic Synthesis:

Quantum computers, due to their reversible nature, lack the full toolkit available in classical synthesis. Unlike classical gates, where output reveals the input state, quantum operations must be reversible. This necessitates the use of reversible building blocks to construct any quantum operation.

Workaround for Non-Reversibility:

Non-reversible gates can be converted into reversible ones by preserving the original inputs and storing the result in a new qubit. For example, an n-controlled X gate allows computing a multi-AND operation between n qubits on a new qubit. Followed by another (multi-)controlled X gate on the same qubit, we can achieve an XOR operation. **XAGs (XOR and AND Graphs):**

Expressions like `(x0 AND x1 AND x2) XOR (x0 AND x1)` are called XAGs. These graphs can be conveniently represented by polynomials over the Boolean algebra (F2).

Reed–Muller Expansion for Truth Tables:

Given a truth table `T` with `n` variables, its Reed–Muller expansion `RMn(x)` is a polynomial generated recursively using the following equation:

RMn(x) = x0 * RMn-1(x) XOR (x0 XOR 1) * RMn-1(x)

Here, `T0` and `T1` represent the cofactors of `T`, which are truth tables obtained by considering entries of `T` where `x0` is either 0 or 1 (see Table 1 in the provided excerpt). This recursion terminates at `n = 0` with `RM0(x)` being 0 or 1 depending on the value of `T`.

Circuit Implementation from Polynomials:

The resulting polynomial can be used to generate a corresponding quantum circuit. Each "summand" in the polynomial translates to a multi-controlled X gate. However,this method still has a lot of room for improvement as multi-controlled gates are computationally very expensive—(8) an n-controlled NOT gate requires (2n 2 - 2n + 1) CNOT gates.

Gray synthesis:-

At the heart of Gray Synthesis lies the ability to control the phase shift of individual computational basis states, which are essentially the input states of our quantum circuit. By manipulating these phase shifts, we can create custom logic functions by applying them to the input register and the output qubit.

To implement this, we'll be using three types of quantum gates: CNOT (controlled-NOT), Hadamard (H), and a custom gate denoted as Tm. While **(author?)** (10) considers directly synthesizing logic functions is an option, the paper discusses limitations associated with this approach, such as the requirement for F2-linearity, which isn't always applicable.

Let's dive into a concrete example to illustrate how Gray Synthesis works:

1. Imagine we want to create a logic function where input states with a 0 in the output qubit experience a phase shift of 0, and those with a 1 in the output qubit experience a phase shift of πT(x) (where T(x) represents the desired truth table function).

2. By applying the Gray Synthesis circuit (denoted by Ugray) to the combined input register and output qubit surrounded by Hadamard gates, we achieve this transformation:

$$\begin{aligned} H_{\text{out}}\, U_{\text{gray}}\, H_{\text{out}}\, |x\rangle|0\rangle &= \frac{1}{\sqrt{2}} H_{\text{out}}\, U_{\text{gray}}\, |x\rangle(|0\rangle + |1\rangle) \\ &= \frac{1}{\sqrt{2}} H_{\text{out}}\, |x\rangle(|0\rangle + \exp(i\pi T(x))|1\rangle) \\ &= \frac{1}{\sqrt{2}} H_{\text{out}}\, |x\rangle \left(|0\rangle + (-1)^{T(x)}|1\rangle\right) \\ &= |x\rangle|T(x)\rangle. \end{aligned}$$

To fully grasp the power of Gray Synthesis, understanding phase synthesis is crucial. The paper

explores the concept of parity operators, which are essentially XOR expressions involving input variables. These operators can be used to "load" specific values onto a qubit using sequences of CNOT gates.

Furthermore, the paper introduces the notion of parity networks. These networks employ RZ gates applied to parity operators to manipulate the phase of input states. The specific phase shift for a given input state is determined by the combination of parity operators traversed and their corresponding RZ gate parameters (θp).

In essence, Gray Synthesis offers a powerful tool for designing logic circuits on quantum computers. It allows us to create custom logic functions by precisely manipulating phase shifts. The detailed explanation of parity operators and networks provides valuable insights into the underlying mechanisms of this method.We can therefore control what kind of phase each input state receives by carefully deciding on how to distribute the phase shifts θ_p.Since each input state x can be distinguished uniquely by examining the set of parity operators that yield 1 when applied to x, we can assign a distinct constellation of phase shifts to each state by iterating through all potential parity operators. The next question is, how to determine the required $\theta = (\theta_x 0\ ,\ \theta_x 1\ ,\ \theta_x 0 \oplus x1\ ..\ .)$ phase shifts for a given sequence of desired overall phase shifts $\phi = (\phi_0,\ \phi_1,...,\ \phi_2 n\ )$? In this context, by progressively documenting which state obtains which phase shift, one can establish a system of linear equations. The resulting matrix is denoted as the *parity matrix D*. The corresponding system of equations therefore yields:

$$\phi = \frac{1}{2} D\theta. \tag{12}$$

This is achieved by ordering the rows according to the natural order of the input states (i.e.000, 001, 010.) and the columns according to an algorithmic solution of the Hamming TSP, which visits all parity operators. The resulting matrix only depends on the number of input qubits. In the case of three input qubits we get:

$$D_3 = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ -1 & -1 & 1 & 1 & -1 & 1 & -1 & 1 \\ 1 & -1 & -1 & 1 & -1 & -1 & 1 & 1 \\ -1 & 1 & -1 & 1 & 1 & -1 & -1 & 1 \\ -1 & -1 & -1 & -1 & 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 & -1 & 1 & -1 & 1 \\ -1 & 1 & 1 & -1 & -1 & -1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & -1 & 1 \end{pmatrix} \tag{13}$$

6.2 Parity operator traversal:-

The space of parity operators plays a crucial role. This space represents all possible manipulations on n-bit systems that flip an even or odd number of bits. Finding an efficient path to visit specific points within this space, known as parity operators, is essential for certain algorithms.

One challenge arises because we only care about a specific subset of parity operators – those with a non-vanishing Hadamard coefficient (a technical term related to quantum mechanics). A straightforward approach like using the Gray code(19), which traverses all possible bit combinations by flipping only one bit at each step, would be inefficient as it visits many unnecessary operators. Traveling Salesman Problem(20) can also be refered.

The heuristic solution using two "salesmen" to navigate the space of parity operators.

1. **First Salesman:** This salesman efficiently visits the required parity operators one by one, ensuring it only flips one bit at a time. This approach resembles the Gray code in terms of minimizing bit flips, but focuses on the specific operators of interest.

2. **Second Salesman:** To ensure a closed loop and return to the starting point, a second salesman is introduced. This salesman mirrors the path of the first salesman but in reverse order. When both salesmen finish their routes, they will meet, forming a complete traversal.

Key Points of the Heuristic:

- Each salesman prioritizes visiting the closest unvisited parity operator within the targeted subset.
- While penalizing large jumps between operators was explored, it did not significantly improve efficiency.

Compared to a naive approach visiting all operators, this method reduces the number of visited operators by roughly 10%. However, it requires more classical resources (resources not specific to quantum computation) for implementation.

6.3 Phase tolerant synthesis:-

Phase tolerant synthesis emerges as a revolutionary approach in optimizing quantum circuit design. It tackles the resource inefficiency associated with traditional synthesis methods by leveraging the inherent tolerance of certain quantum operations to phase errors. This section delves deeper into the concept, incorporating relevant formulas to illustrate its power.

The Flaw in Conventional Methods:

Existing synthesis methods, like Gray and PPRM, aim to achieve a specific state within the circuit. This state features a label register, representing potential solutions, in a uniform superposition – all possibilities existing with equal probability (represented by the summation symbol Σ). This state is described by the formula:

$$|s\rangle = \sqrt{(1/2\hat{n})} * \Sigma(\ |x\rangle\ )\ (1)$$

where:

- $|s\rangle$ represents the overall state of the circuit
- n is the number of qubits in the label register
- $|x\rangle$ represents each possible state of the register (combination of 0s and 1s)

To grasp the synthesis process involving garbage-phases, it's important to recognize that in this method, the logical data is solely encoded in the phase disparities between the 0 and 1 states of the output qubit. Following the principles applied , we arrive at:

$$\begin{aligned} H\left(\exp\left(i\phi_0\right)|0\rangle + \exp\left(i\phi_1\right)|1\rangle\right) &= \exp\left(i\phi_0\right) H\left(|0\rangle + \exp\left(i\left(\phi_1 - \phi_0\right)\right)|1\rangle\right) \\ &= \begin{cases} \exp\left(i\phi_0\right)|0\rangle \text{ if } \phi_1 - \phi_0 = 0 \\ \exp\left(i\phi_0\right)|1\rangle \text{ if } \phi_1 - \phi_0 = \pi \end{cases} \end{aligned} \tag{14}$$

To formalize this concept further, let's consider the scenario where we are applying Gray synthesis to the state of uniform superposition $|s\rangle$:

$$|s\rangle = \frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n} |x\rangle. \tag{15}$$

We factor out the output qubit:

$$|s\rangle = \frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^{n-1}} (|0\rangle + |1\rangle)|x\rangle \tag{16}$$

The 'first step' in this process of Gray synthesis is synthesizing phases $\phi_{(0,x)}$ and $\phi_{(1,x)}$ on the output qubit:

$$U\text{Step}(s) = \frac{1}{2\sqrt{2^n}} \left(\exp(i0x) + \exp(i\phi(1,x))\right)|x| = |\psi(x)|. \tag{17}$$

The second step would now be to synthesize a phase on the remaining qubits which could be seen as 'correcting' the garbage phase X_x. This however does not change the relative phase of the $|0\rangle$ and $|1\rangle$ state of the output qubit:

$$U_{\text{gray}}^{\text{Step 2}}|\psi\rangle = \frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^{n-1}} \exp\left(-i\chi_x\right)\left(\exp\left(i\phi_{(0,x)}\right)|0\rangle + \exp\left(i\phi_{(1,x)}\right)|1\rangle\right)|x\rangle \tag{18}$$

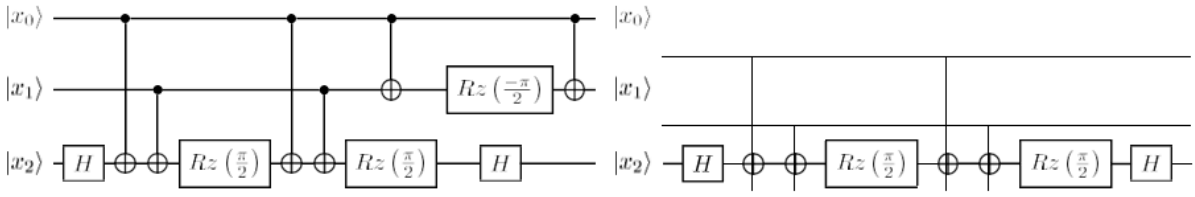
$$\text{UStep2}\left(\frac{2-1}{2\sqrt{\pi}}\right) = \exp(-i\cdot xx)\left(|0\rangle + |1\rangle\right)\Big|_x = |\psi(x)\rangle. \tag{19}$$

The important point of phase tolerant synthesis is that the *difference* $\phi_{(0,x)}$ - $\phi_{(1,x)}$ determines the result of the logic output state:

$$\phi_{(0,x)} - \phi_{(1,x)} = \begin{cases} 0 & \text{if } T(x) = 0 \\ \pi & \text{if } T(x) = 1. \end{cases} \tag{20}$$

7 CSE synthesis:-

In this section, we present further progress beyond state-of-the-art from our research activities—the implementation of quantum logic synthesis under the consideration of restrictions in a real world setup. Phase tolerant Gray synthesis may be very resource friendly to both classical and quantum resources, however scaling on real devices is still challenging: Taking a look at equation we see that, because the only values of the entries of ϕ that can appear in the scenario of logic synthesis are $\pm\ \pi$. Therefore the values of the entries of ϕ are integer multiples of $\pm\ \pi$. This implies that if we want to encode an array with N entries,



CSE synthesis is a technique used to efficiently implement the oracle gate, which is a crucial component of the algorithm. The oracle gate marks the target state(s) in the search space, allowing Grover's algorithm to amplify the probability amplitudes of these states.

The oracle gate is implemented using CSE synthesis to perform a conditional phase shift on the target state(s). The CSE synthesis technique involves several steps:

1. Initialization: Initialize a quantum register with qubits representing the search space. These qubits are typically put into a superposition of all possible states.

2. Oracle Construction: Construct an oracle gate that flips the phase of the target state(s) while leaving all other states unchanged. This oracle gate can be represented as a diagonal matrix with phases flipped for the target state(s).

3. Conditional Shifted Expansion (CSE): Apply the oracle gate conditionally based on the current state of the qubits. This involves using controlled operations to ensure that the oracle gate is applied only when the qubits are in the target state(s). The CSE technique efficiently expands the oracle gate to act on the entire search space while avoiding unnecessary operations on non-target states.

4. Grover Iterations: Repeat the application of the oracle gate followed by the inversion about the mean (diffusion operator) for a certain number of iterations. This process amplifies the probability amplitudes of the target state(s) while suppressing the amplitudes of other states.

5. Measurement: Finally, measure the qubits to collapse the superposition and obtain the solution(s) with high probability.

CSE synthesis plays a crucial role in optimizing the efficiency of Grover's algorithm by minimizing the computational resources required to implement the oracle gate. This helps in achieving the quadratic speedup provided by Grover's algorithm for unstructured search problems(14).

References

[1] F. Arute, K. Arya, R. Babbush, D. Bacon, J. C. Bardin, R. Barends, R. Biswas, S. Boixo, F. G. Brandao, D. A. Buell *et al.*, "Quantum supremacy using a programmable superconducting processor," *Nature*, vol. 574, no. 7779, pp. 505–510, 2019.

[2] G. Sun, S. Su, and M. Xu, "Quantum algorithm for polynomial root finding problem," in *2014 Tenth International Conference on Computational Intelligence and Security*. IEEE, 2014, pp. 469–473.

[3] A. Gilliam, S. Woerner, and C. Gonciulea, "Grover adaptive search for constrained polynomial binary optimization," *Quantum*, vol. 5, p. 428, 2021.

[4] I. Chakrabarty, S. Khan, and V. Singh, "Dynamic grover search: Applications in recommendation systems and optimization problems," *Quantum Information Processing*, vol. 16, pp. 1–21, 2017.

[5] W. P. Baritompa, D. W. Bulger, and G. R. Wood, "Grover's quantum algorithm applied to global optimization," *SIAM Journal on Optimization*, vol. 15, no. 4, pp. 1170–1184, 2005.

[6] S. E. Borujeni, R. Harikrishnakumar, and S. Nannapaneni, "Quantum grover search-based optimization for innovative material discovery," in *2019 IEEE International Conference on Big Data (Big Data)*. IEEE, 2019, pp. 4486–4489.

[7] G. Brassard, P. Høyer, and A. Tapp, "Quantum cryptanalysis of hash and claw-free functions," in *LATIN'98: Theoretical Informatics: Third Latin American Symposium Campinas, Brazil, April 20–24, 1998 Proceedings 3*. Springer, 1998, pp. 163–169.

[8] A. Abdollahi, M. Saeedi, and M. Pedram, "Reversible logic synthesis by quantum rotation gates," *arXiv preprint arXiv:1302.5382*, 2013.

[9] A. N. Al-Rabadi, *Reversible logic synthesis: from fundamentals to quantum computing*. Springer Science & Business Media, 2004.

[10] M. Amy, P. Azimzadeh, and M. Mosca, "On the controlled-not complexity of controlled-not–phase circuits," *Quantum Science and Technology*, vol. 4, no. 1, p. 015002, 2018.

[11] S. Anis, "Qiskit: An open-source framework for quantum computing," *(No Title)*, 2021.

[12] G. Brassard, P. Hoyer, M. Mosca, and A. Tapp, "Quantum amplitude amplification and estimation," *Contemporary Mathematics*, vol. 305, pp. 53–74, 2002.

[13] G. Chen, S. A. Fulling, H. Lee, and M. O. Scully, "Grover's algorithm for multiobject search in quantum computing," in *Directions in Quantum Optics: A Collection of Papers Dedicated to the Memory of Dan Walls Including Papers Presented at the TAMU-ONR Workshop Held at Jackson, Wyoming, USA, 26–30 July 1999*. Springer, 2001, pp. 165–175.

[14] H. M. Eckhoff, K. Bech, G. Bouma, K. Eide, D. Haug, O. E. Haugen, and M. Jøhndal, "The PROIEL treebank family: A standard for early attestations of Indo-European languages," *Language Resources and Evaluation*, vol. 52, no. 1, p. 29–65, 2018.

[15] C. Fabricius-Hansen and D. T. T. Haug, "Co-eventive adjuncts: Main issues and clarifications," in *Big events, small clauses*, C. Fabricius-Hansen and D. T. T. Haug, Eds. Berlin-Boston: De Gruyter, 2012, pp. 21–54.

[16] Fino and Algazi, "Unified matrix treatment of the fast walsh-hadamard transform," *IEEE Transactions on Computers*, vol. 100, no. 11, pp. 1142–1146, 1976.

[17] I.-D. Gheorghe-Pop, N. Tcholtchev, T. Ritter, and M. Hauswirth, "Computer scientist's and programmer's view on quantum algorithms: mapping functions' apis and inputs to oracles," in *Intelligent Computing: Proceedings of the 2021 Computing Conference, Volume 1*. Springer, 2022, pp. 188–203.

[18] P. Porwik, "Efficient calculation of the reed-muller form by means of the walsh transform," 2002.

[19] A. D. R. Kulandai, J. Rose, and T. Schwarz, "Balanced gray codes for reduction of bit-flips in phase change memories," in *Modelling, Analysis, and Simulation of Computer and Telecommunication Systems: 28th International Symposium, MASCOTS 2020, Nice, France, November 17–19, 2020, Revised Selected Papers 28*. Springer, 2021, pp. 159–171.

[20] C. E. Miller, A. W. Tucker, and R. A. Zemlin, "Integer programming formulation of traveling salesman problems," *Journal of the ACM (JACM)*, vol. 7, no. 4, pp. 326–329, 1960.