Quantum Maze Solver implementing Grover's Algorithm

Federico Bravetti
federico.bravetti2@studio.unibo.it

Michele Ravaioli
michele.ravaioli3@studio.unibo.it

Matteo Bambini
matteo.bambini@studio.unibo.it

Valerio Giannini
valerio.giannini@studio.unibo.it

Abstract

This paper presents a quantum maze solver implementation using Grover's algorithm for quadratic speedup over classical approaches. By mapping the maze into a graph, the problem was translated into a generic path finding problem making the solution applicable to a wider range of problems. Simulation on 2x2 maze achieves 98.60% success probability with optimal Grover iterations. The solution highlights quantum computing's potential for path finding problems while addressing the scalability and qubit allocation challenges inherent to quantum architectures.

Introduction

Maze solving represents a fundamental class of graph traversal problems with extensive applications in robotics, autonomous navigation, and strategic path planning. In the classical computing paradigm, solving mazes requires exploring all possible paths sequentially; the optimal computational complexity for path-finding problem is $O(N+E)$ where N is the number of nodes/cells and E are the number of edges, that cost becomes $O(N)$ in case of maze (or sparse graph in general). Quantum computing offers an intriguing alternative through Grover's algorithm, achieving a theoretical quadratic speedup, reducing this complexity.

This project explores how to leverage Grover's algorithm to find valid paths in a maze, encoded as a quantum search space. Our implementation covers:

- Translating the maze into a graph structure
- Binary encoding of nodes into qubit registers
- Quantum oracle design for path validation
- Diffuser construction for amplitude amplification
- Simulation and comparison with classical BFS

Problem Formulation

The given solution works by representing the maze as a undirected graph, containing one starting node, one ending node, and only one unique path connecting the two nodes, so we deal with perfect mazes.

Given a directed graph $G = (V, E)$ with:

- V: Set of nodes (maze junctions);
- $E \subseteq V \times V$: Valid transitions between nodes;
- $s \in V$: Fixed start node (entrance);
- $g \in V$: Fixed goal node (exit);
- $n := |V|$: Number of nodes;
- $e := |E|$: Number of edges;
- $l := |P|$: Solution path's length;

the algorithm should find the unique valid path $P = [v_0 = s, v_1, ..., v_{l-2}, v_{l-1} = g]$ of length l such that:
$(v_i, v_{i+1}) \in E$ for all $0 \leq i < l-1$.

Grover's Algorithm

Grover's algorithm, first described in a 1996 paper, allows finding a specific element in a table of N unsorted elements using $O(\sqrt{N})$ operations. In comparison, a classical computer requires a computational complexity of $O(N)$.

The algorithm's operation is based on an oracle, which enables a general description and geometric interpretation.

A. Oracle Definition

We search for M solutions within a space of N elements, where $1 \leq M \leq N$.

The function $f(x)$ is characterized as follows:

$$f(x) = \begin{cases} 1 & \text{if } x \text{ is a solution,} \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

The oracle designates solutions through phase modification:

$$|x\rangle \xrightarrow{U_f} (-1)^{f(x)}|x\rangle. \tag{2}$$

Given M solutions, the oracle requires $O\left(\sqrt{\frac{N}{M}}\right)$ iterations.

B. *Diffuser Definition*

The Diffusion operator is defined as:

$$D = 2|\psi\rangle\langle\psi| - I_N \tag{3}$$

where I_N is the identity on N qubits and ψ represent the uniform superposition:

$$|\psi\rangle = \frac{1}{\sqrt{N}} \sum_x |x\rangle \tag{4}$$

The effect consists of amplifying the designated solution amplitude while decreasing the others:

$$\sum_x \alpha_x |x\rangle \xrightarrow{D} \sum_x (2\mu - \alpha_x)|x\rangle \tag{5}$$

with μ the mean of the state amplitudes:

$$|\mu\rangle = \frac{1}{\sqrt{N}} \sum_x |\alpha_x\rangle \tag{6}$$

C. *Algorithm Iterations*

The Grover iteration comprises the following steps:

- The circuit employs a register of n qubits, initialized in state $|0\rangle^{\otimes n}$
- The Hadamard transform $H^{\otimes n}$ generates the superposition.
- Apply $G = DU_f$ for the required number of iterations k.

The overall effect can be expressed as:

$$(\prod_k G)H^{\otimes n}. \tag{7}$$

D. *Geometric Interpretation*

We can reduce our n-dimensional Hilbert space into a two-dimensional one using the following basis:

$$|\alpha\rangle = \frac{1}{\sqrt{N-M}} \sum_x^{\prime\prime} |x\rangle, \quad |\beta\rangle = \frac{1}{\sqrt{M}} \sum_x^{\prime} |x\rangle, \tag{8}$$

where $|\alpha\rangle$ and $|\beta\rangle$ represent non-solution and solution states, respectively. The uniform superposition state can be written as follow:

$$|\psi\rangle = \sqrt{\frac{N-M}{N}}|\alpha\rangle + \sqrt{\frac{M}{N}}|\beta\rangle. \tag{9}$$

The oracle operator U_f reflects around $|\alpha\rangle$, while the diffusion operator D reflects around $|\psi\rangle$.

The overall application of G, as shown in fig. 1 result in a rotation by angle θ given:

$$\cos\frac{\theta}{2} = \sqrt{\frac{N-M}{N}} \tag{10}$$

$$\sin\frac{\theta}{2} = \sqrt{\frac{M}{N}} \tag{11}$$

And after k iterations we have:

$$G^k|\psi\rangle = \cos\left(\frac{2k+1}{2}\theta\right)|\alpha\rangle + \sin\left(\frac{2k+1}{2}\theta\right)|\beta\rangle. \tag{12}$$

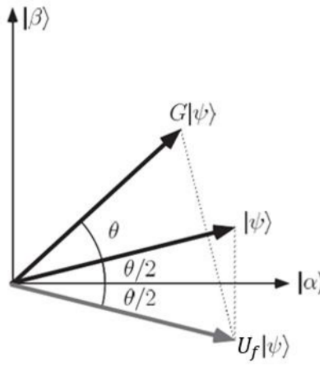

Figure 1. Effect of the rotation

Each iteration rotates the state closer to $|\beta\rangle$, increasing the probability of finding a solution. The optimal number of iterations is:

$$K_{\text{opt}} \approx \frac{\pi}{4}\sqrt{\frac{N}{M}}. \tag{13}$$

Exceeding this number reduces success probability, though it eventually increases again due to the rotation's periodic nature.

DESIGN

Using Grover's algorithm, the problem reduces to the construction of the oracle U_f, which flips the phase of the correct input configuration. We consider as input the sequence of nodes of a possible solution; since its length is initially unknown, the chosen sequence is the one that passes through all the maze's cells, so we consider:

- $l := n - 1$.

To also evaluate shorter solutions, those will terminate with a sequence of the last node; this is possible by manually adding a self-loop on the exit node.
Given x a set of nodes and l the path length, we define the function f implemented by the oracle as:

$$f(x) = (x_0 = s) \land (x_{l-1} = g) \land fec(x) \land ftbc(x) \tag{14}$$

The function fec, referred to as the *full edge check function*, verifies that the entire sequence constitutes a valid path in the graph by checking whether each consecutive node pair in x belongs to the edge set E.

The function $ftbc$, referred to as the *full turn back check function*, verifies that the path didn't contain the same node twice unless is the exit node.

By properly implementing these functions, the oracle U_f can be constructed, enabling the application of Grover's algorithm.

Implementation

For implementation purpose the maze nodes are encoded using the binary representation of node IDs so we define:

- $b := \lceil \log_2(n) \rceil$ — the number of qubits required to represent a node;

For better clarity in the following sections, we explain the implementation referring to an example 2×2 maze, but it can be, of course, extended to arbitrary mazes.

E. Oracle Implementation

As said before, the oracle is composed of two main functions: the full edge check function (fec) and the full turn back check ($ftbc$).

The fec applies the edge check circuits (ec) to each couple of nodes in the input path except for the first and the last, which contain a special check to verify that they are respectively the start and the end cells of the maze. The ec circuits check if a pair of nodes matches an existing edge (as shown in fig. 2) and each of them stores the result into its own ancilla. So in total for this control are required $n-1$ ancillas. A single edge check is done as follows:

- Applies X gates to input qubits where the bit value should be 0
- Uses *multi-controlled-X* gate to check if all input qubits are $|1\rangle$ (which means that the edge is valid) and store the result into an ancilla
- Re-applies X gates to restore input values to original state

Due to the structure of the ec circuit, it's guaranteed that the ancilla is flipped at most one time.

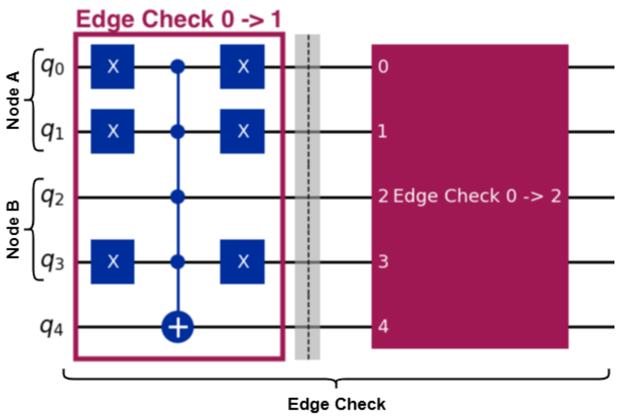

Figure 2. Edge Check Circuit for a 2x2 maze

Regarding the $ftbc$ function, it applies the turn back check (tbc) circuit to each set of 3 nodes in the input path. The tbc circuit checks that the first and the third nodes are different or, in case they're equal, it checks that the first two nodes are the same as the last node, allowing looping on the last node when it is reached. Each tbc circuit, as shown in fig 3, stores its result into an ancilla. So in total for this control are required $n-2$ ancillas.

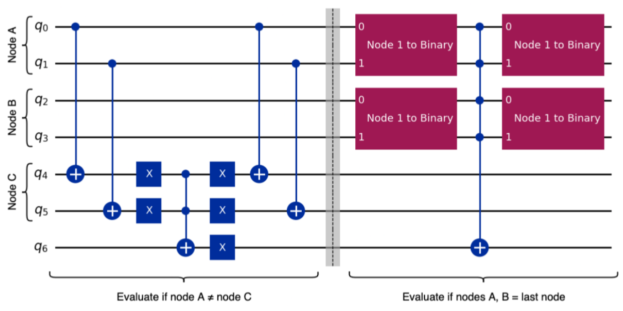

Figure 3. Turn Back Check Circuit for a 2x2 maze

After all the checks are done, a *multi-controlled-Z* gate is applied to all ancillas; this gate flips the phase if a valid path is recognized so when all ancillas are set to $|1\rangle$. The resulting oracle is reported in fig. 4; this circuit has also a block called *Path Check_dg*, which performs the garbage collection operation, needed in order to avoid entanglement interference issues.

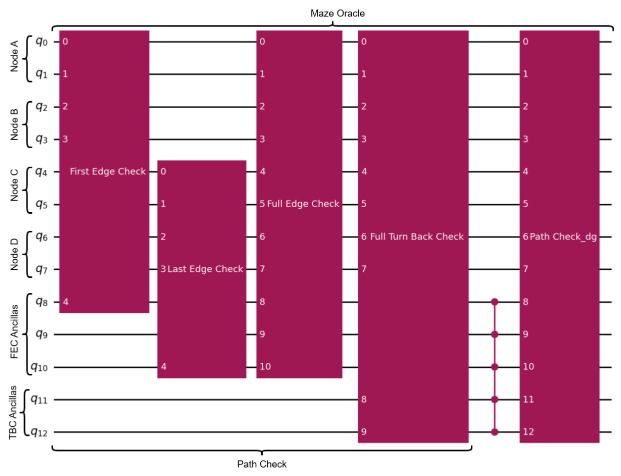

Figure 4. Maze Oracle for a 2x2 maze

F. Diffuser Implementation

Considering the definition of the diffusion operator (Eq. 3), and observing that

$$|\psi\rangle = H^{\otimes N}|0\rangle, \tag{15}$$

and that

$$I_N = H^{\otimes N} I_N H^{\otimes N}, \tag{16}$$

we can express the diffusion operator as:

$$D = H^{\otimes N}\left(2|0\rangle\langle 0| - I_N\right) H^{\otimes N}. \tag{17}$$

Moreover, the operator inside the parentheses can be implemented as:

$$2|0\rangle\langle 0| - I_N = X^{\otimes N} CZ X^{\otimes N}. \tag{18}$$

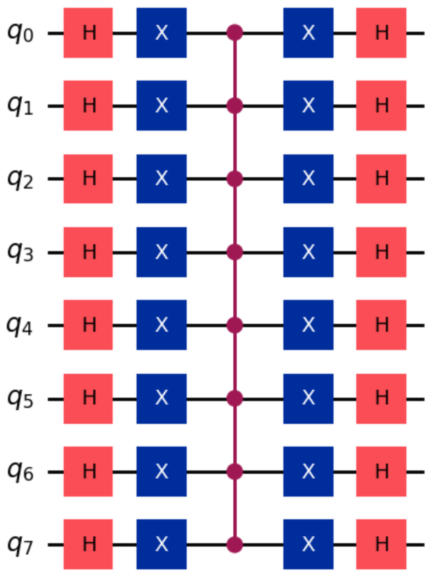

Figure 5. Diffuser for 8 data qubits

G. Circuit Assembly

Here we describe how the example 2×2 maze reported in fig. 6 is implemented:

- Quibits for a node $\rightarrow b = \lceil \log_2(4) \rceil = 2$;
- Max path length $\rightarrow l = n - 1 = 3$;
- Quibits for the path nodes $L \rightarrow b \cdot (l+1) = 8$
- Ancilla for $fec \rightarrow l = 3$
- Ancilla for $ftbc \rightarrow l - 1 = 2$

So the overall circuit will result in:

- Total qubits: $L + fec + ftbc = 13$ (8 data + 5 ancilla);
- Initialization: $H^{\otimes 8}$ on data qubits;
- The oracle and the diffuser will be repeated K_{opt} times.

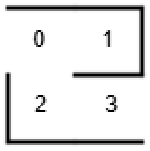

Figure 6. Example maze

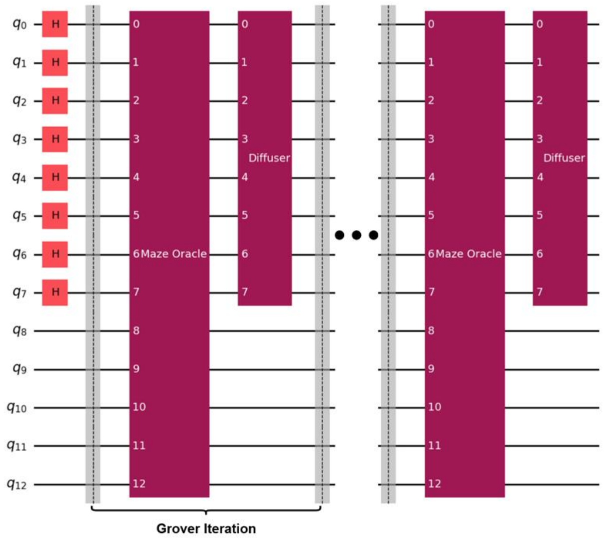

Figure 7. Complete Circuit for the example maze

Complexity Analysis

We can analyze the performance of the algorithm in terms of time and space complexity. Specifically, we examine: the execution time complexity, the circuit construction time, the number of qubits required, and the total number of gates used.

Execution Time Complexity

The Grover-based quantum maze solver algorithm has a time complexity of:

$$\mathcal{O}\left(e \cdot \sqrt{2^L}\right) = \mathcal{O}\left(e \cdot \sqrt{n^l}\right) \tag{19}$$

This complexity reflects two key factors:

- A linear dependence on the number of edges e, due to edge validation checks;
- An exponential dependence on the path length L.

Qubit Complexity

The total number of qubits required by the circuit is:

$$L + 2l - 1 \tag{20}$$

Gate Complexity

The overall number of quantum gates in the circuit is:

$$\mathcal{O}\left((e + l) \cdot b \cdot \sqrt{n^l}\right) \tag{21}$$

This accounts for the repetition of Grover's operator across $\mathcal{O}(\sqrt{n^l})$ iterations, each involving circuits that scale with $\mathcal{O}(e)$.

Circuit Construction Complexity

Constructing the circuit is also computationally non-trivial. The time complexity of the circuit generation process is:

$$\mathcal{O}\left(e + lb + \sqrt{n^l}\right) \tag{22}$$

This complexity is lower than the execution time, since some optimizations can be applied: edge-check and turn-back subcircuits need to be generated only once and be reused across the circuit.

Overall the construction cost remains efficiently manageable through circuit modularity and reuse but the execution time remain dominant despite the Grover complexity advantage.

Results

Simulations were performed using Qiskit's AerSimulator over the example maze. To do so, the quantum circuit was first transpiled to match the simulator's requirements. Then, all qubits encoding the path were measured over 10,000 shots. Finally, the resulting bitstrings were decoded back into path sequences by mapping them to the corresponding node IDs. The measurement results of the simulation run on the example maze (measuring the 8 data qubits) are shown in the table below:

Table I
Measurement Outcomes

Path	**Occurrences**
$0 \to 2 \to 3$	9,860 (98.6%)
Other paths	140 (1.4%)

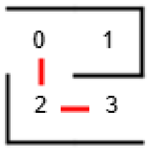

Figure 8. Solved Example maze

After several executions, the simulation showed that the correct solution path was amplified on average with 98.60% probability.

Conclusion

This project presents a proof of concept for applying Grover's quantum algorithm to path-finding problems, by modeling the maze as a graph traversal task and encoding potential solutions into quantum states. We successfully implemented and validated a Grover-based solver. Simulations confirm the correctness of the method on small graphs, laying the groundwork for scaling to larger graph-based path-finding problems as qubit availability increases.

One key advantage of this approach is its potential to generalize to arbitrary graphs, beyond simple mazes. In particular, the algorithm continues to work correctly in the case of acyclic graphs and is capable of identifying multiple valid solutions when they exist. Specifying the expected number of solutions further improves accuracy. However, to support more complex graph structures—such as those containing cycles or self-loops—the oracle would need to be adapted accordingly. The main limitation of the current approach is that the number of qubits required to build the circuit grows quasi-linearly (i.e., $O(n \log n)$) with the number of nodes, which may become inefficient for large-scale problems.

An alternative strategy, more tailored to maze solving, involves representing a path as a sequence of moves. Since the set of possible directions is limited (typically four: up, down, left, right), each move could be encoded using only 2 qubits, and the total number of qubits required to represent a path would therefore grow linearly with the path length. However, while this representation is more compact, it may require a more complex oracle, whose construction and evaluation cost would need to be carefully analyzed.