Quantum Computing
A Practical Perspective

Marco Venere

marco.venere@polimi.it

November 21st, 2024
Politecnico di Milano

Agenda

- Lecture 1 → Theory Recap on Quantum Computing
- Lecture 2 → Initial Setup and First Experiments
- Lecture 3 → Grover's Algorithm
- Lecture 4 → Combinatorial Optimization
- Lecture 5 → VQE, QNN, QMC
- Lecture 6 → Quantum Error Correction & Mitigation – Projects Presentation

Grover's Algorithm

Grover's Algorithm allows for solving the search problem with a quadratic speedup w.r.t. classical computation.

Imagine you have a set of N items that you want to visit to look for something. At most, how many visits do you need to perform?

Classical computing: $O(N)$ visits.

Quantum computing: $O\left(\sqrt{N}\right)$ visits, thanks to Grover's Algorithm.

It's a quadratic speedup!

The Search Problem

Let's try to express this problem in a more formal way.

Search Problem: Given a function $f: \{0,1\}^n \rightarrow \{0,1\}$, the search problem is to find a $y \in \{0,1\}^n$ such that $f(y) = 1$, with the promise that exactly one or at least one solution exists.

Meaning: we have n bitstrings; a function f receives the bitstring and gives 1 as output if the bitstring is the one we are looking for, otherwise it returns 0.

The implementation of f provides the criterion that we use to decide whether the input bitstring is the one we are looking for. For the algorithm, we don't need to know how it is implemented: it is a **black box**.

The Oracle

The oracle implements a generic Boolean function:

$$f(i) = \begin{cases} 0, & \text{if } i \notin \{w_j\} \\ 1, & \text{if } i \in \{w_j\} \end{cases}$$

The domain of f can be a generic set, representing an unstructured database, or a truth table of a boolean function. i is the index of the i-th element, and $\{w_j\}$ is the set of the elements respecting the search criterion. Any kind of data can be considered as input for such an oracle, as long as it is convertible into a proper bitstring, by the usage of a **labeling function**.

The Labeling Function

The labeling function is defined as:

$$l : D \to \mathbb{F}_2^k$$
$$e \to l(e).$$

where k is the label size and D is the database of entries.

A simple way to implement the l function is to compute the hash of the entry.

Encoding a Database

We can encode an entire database in an oracle.

Algorithm 1 Database encoder

Input: Iterable database D, labeling function
Output: Encoding circuit U_D

```
list bitstrings = []
for e in D do
    bitstrings.append(Label(e))
end for
TruthTable tt = TruthTable(bitstrings)
return QuantumLogicSynthesis(tt)
```

The Algorithm

The algorithm proceeds in three steps:

1. Phase Inversion

2. Inversion Around The Mean

3. Iteration of the Diffuser

The Algorithm: Phase Inversion

In the phase inversion, we query the oracle and encode its output in the phase of the qubit.

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

This means that every basis state in superposition gets a +1 or -1, based on the value of the function at that specific basis state.

The Algorithm: Inversion Around The Mean

This stage defines a diffuser operator. This operator is defined as:

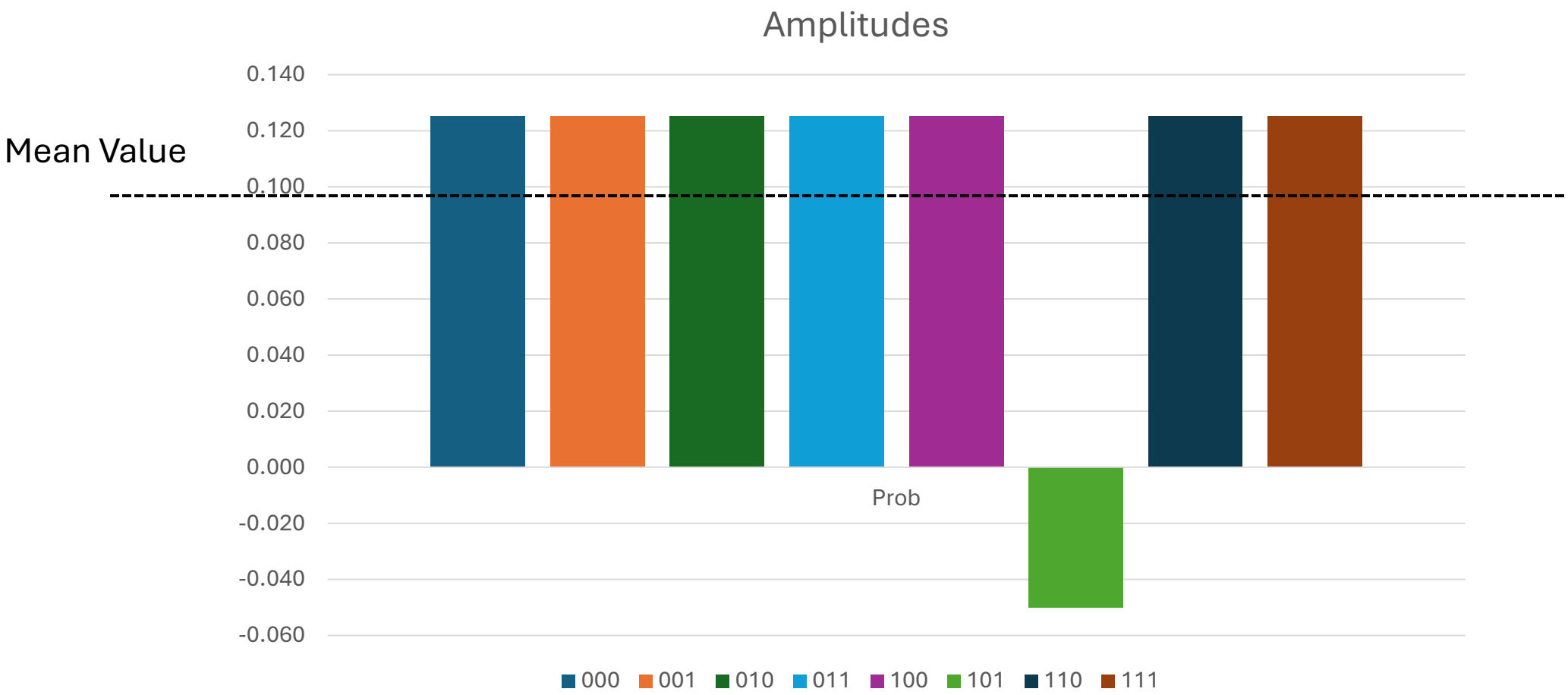
$$I - 2\,|e\rangle\langle e|$$

where e is a linear superposition of all states.

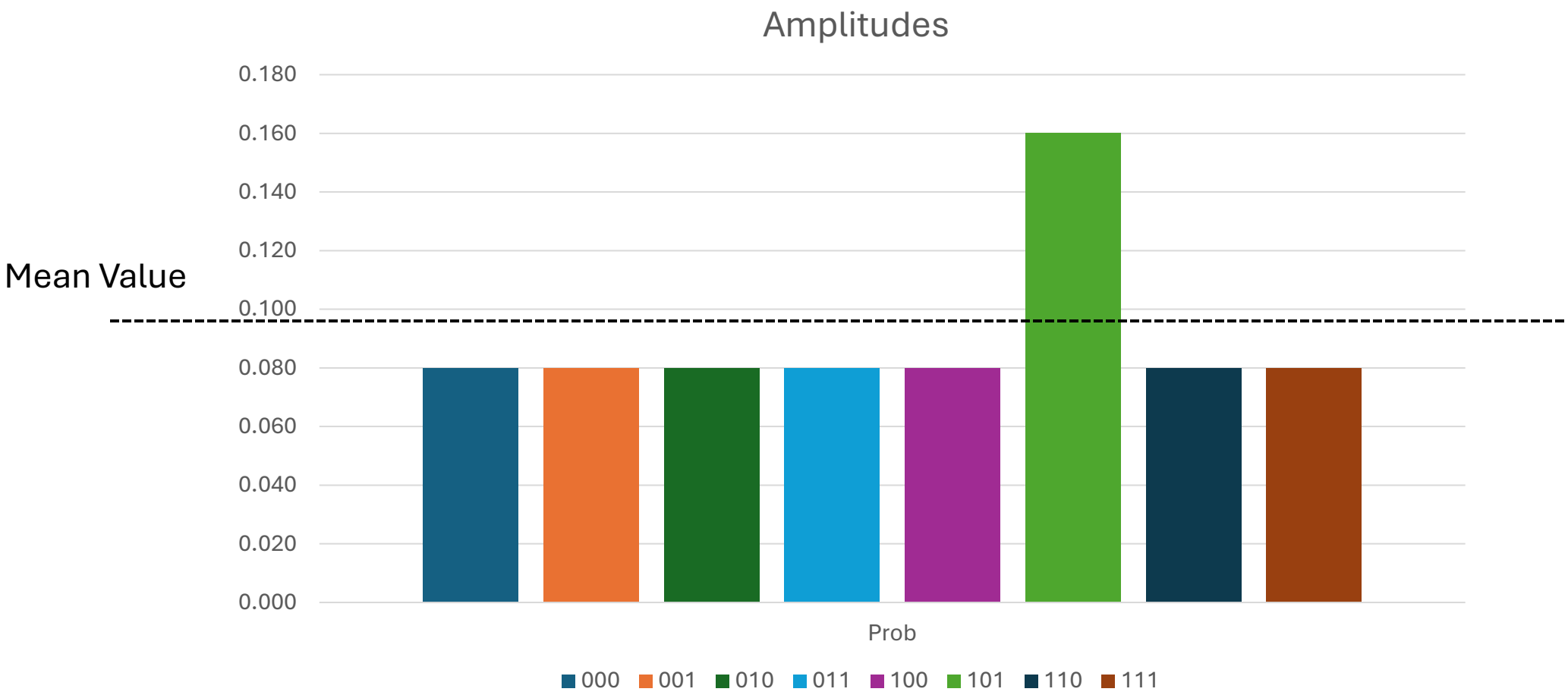
The objective of this operator is to invert the phase of the basis states in the superposition, while increasing their absolute amplitude values.

Doing so, the probability of measuring the state corresponding to the solution to the problem increases.

The Algorithm: Inversion Around The Mean



The Algorithm: Inversion Around The Mean



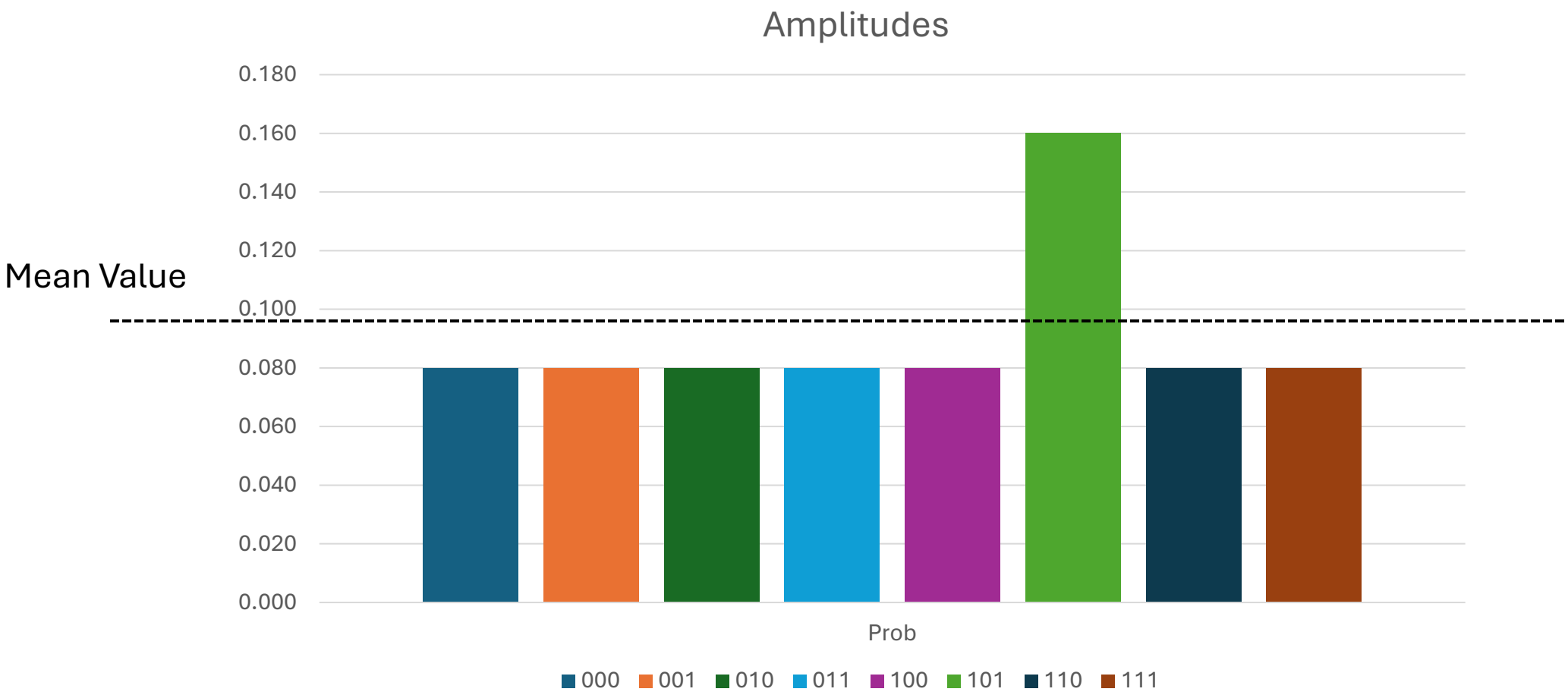
The Algorithm: Iteration Of The Diffuser

We iterate the diffuser operator a number of times, in order to gradually increase the probability of finding the correct solution.

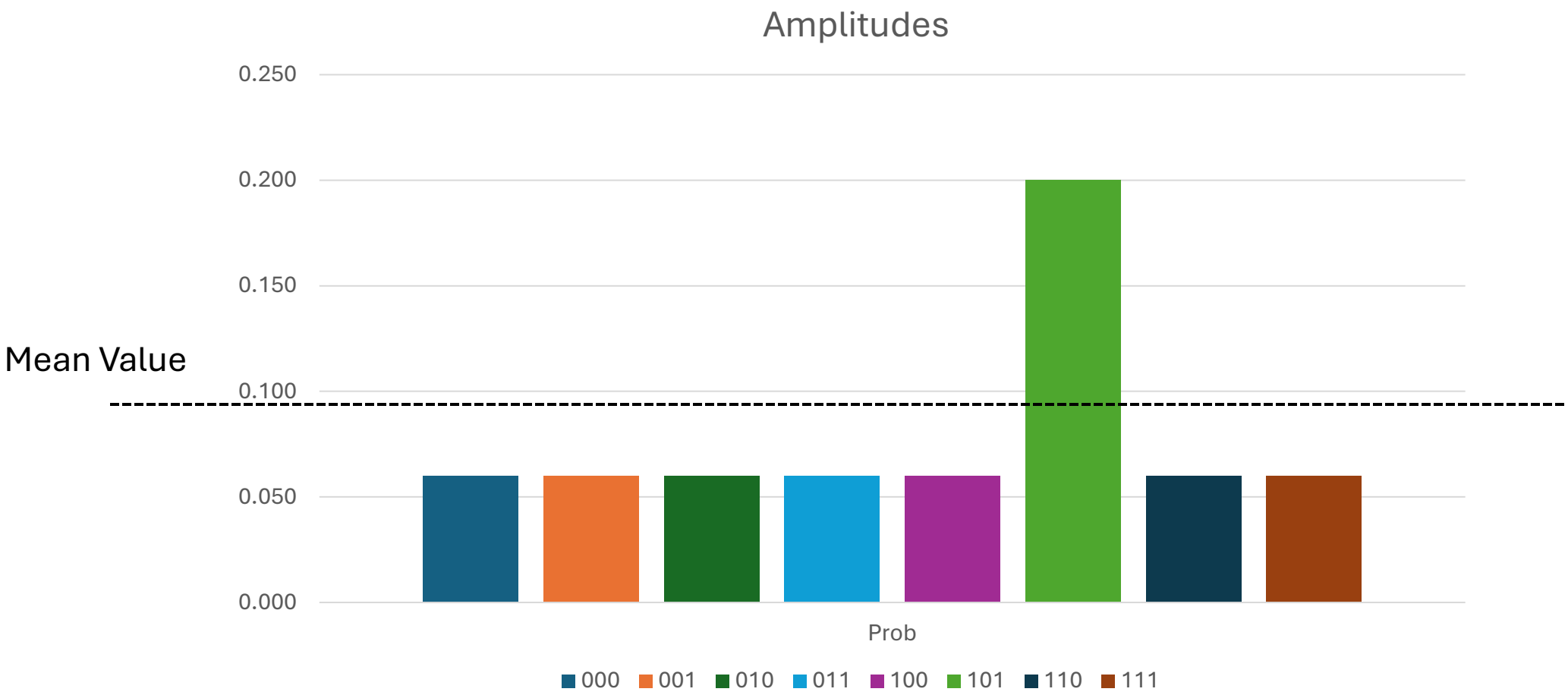
In order to make such a probability close to 1, we need to iterate the diffuser $O\left(\sqrt{2^n}\right)$ times.

This means that we have a quadratic speedup in the number of queries that we perform onto our oracle!

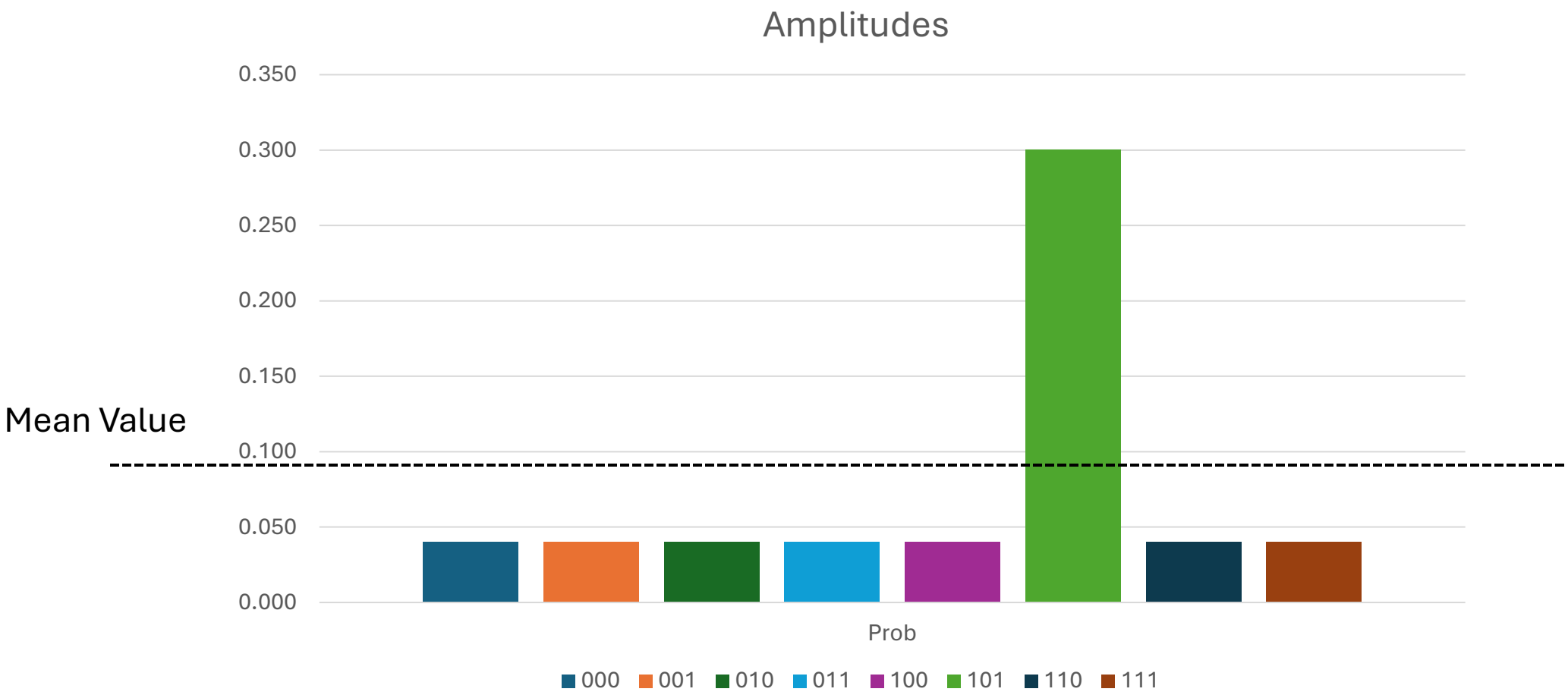
The Algorithm: Iteration #1



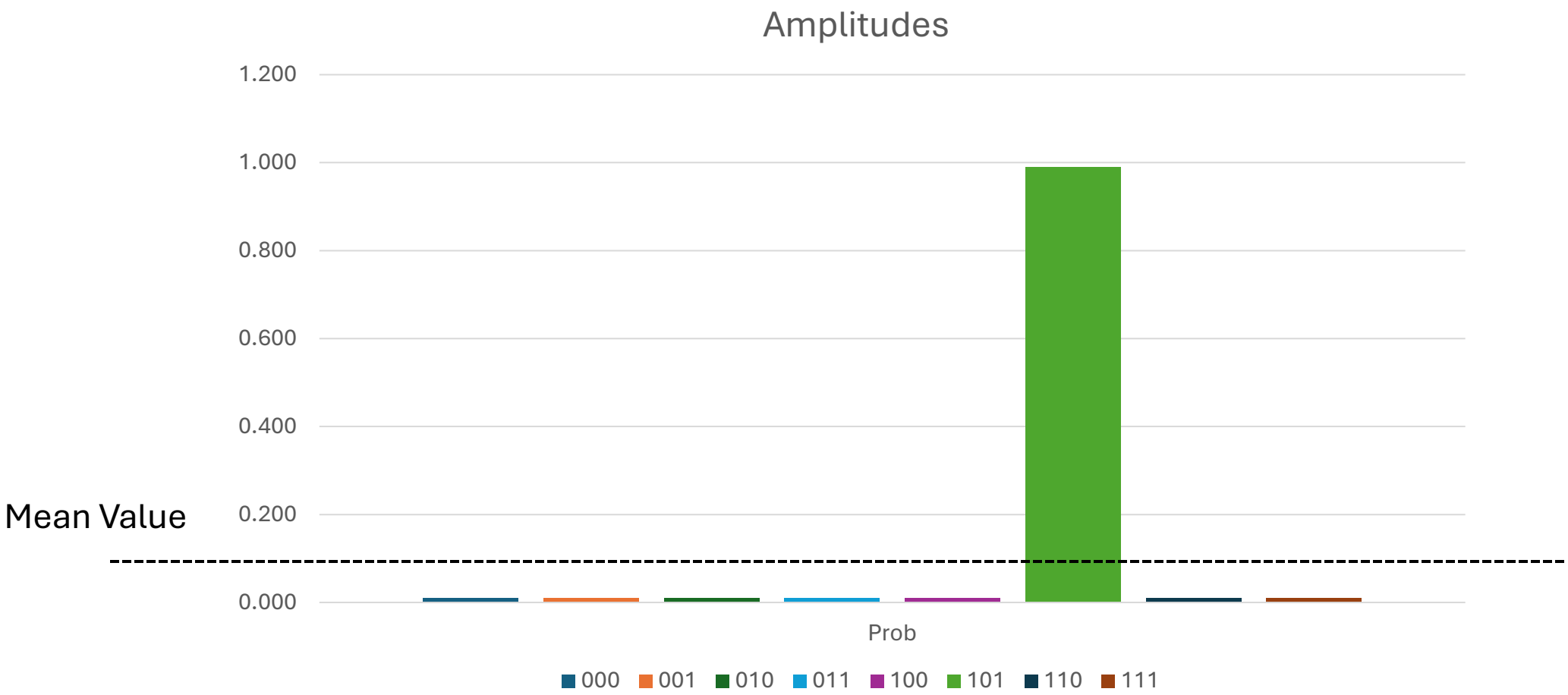
The Algorithm: Iteration #2



The Algorithm: Iteration #3



The Algorithm: Iteration # $O\left(\sqrt{2^n}\right)$



Let’s experiment with MATLAB!

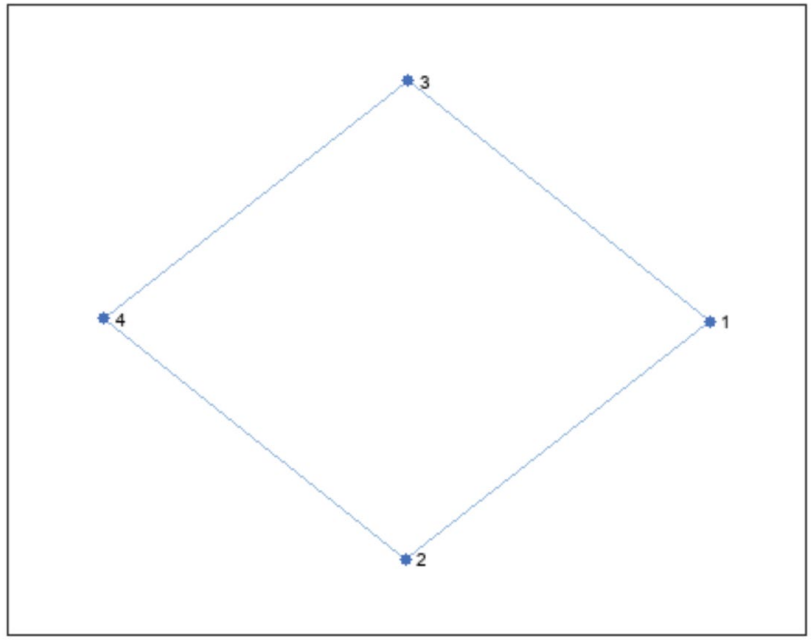
We can now see some code on MATLAB…

Let’s experiment with MATLAB!

Let’s use Grover’s Algorithm to solve the Graph Coloring Problem!

Problem: given a graph, such that no two adjacent vertices are of the same color.

Example Instance:

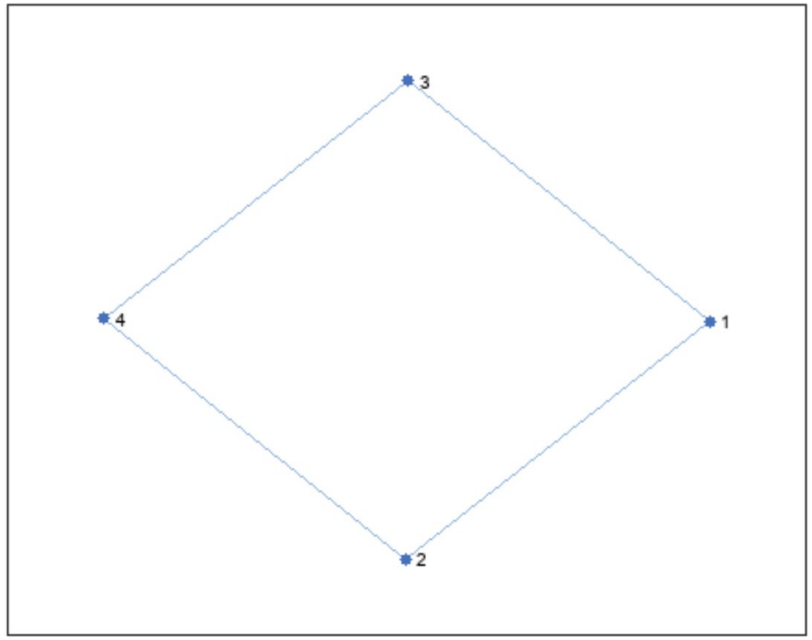


Let’s experiment with MATLAB!

Let’s use Grover’s Algorithm to solve the Graph Coloring Problem!

Problem: given a graph, such that no two adjacent vertices are of the same color.

Example Instance:



Let’s experiment with MATLAB!

How can we approach this problem with Grover?

We can define an oracle that receives a bitstring and verifies whether it is a correct coloring or not.

Grover’s Algorithm will then explore all possible bitstrings and identify one which is correct, according to the oracle. This will produce the solution to the problem!

Let’s try!

Advanced Queries with Grover

Up to now, we assumed that the oracle simply looks for a specific bitstring…

But what if we are only interested in a bitstring which is similar to a given one?

Similarity Search is a specific kind of search where the oracle evaluates a similarity metric between the input and the bitstrings.

The Hamming Distance as Similarity Metric

For example, let's assume the oracle wants to evaluate the Hamming distance as similarity metric.

Assumption: we evaluate the Hamming Distance between bitstrings. In case we are encoding a database of entries in the oracle, such a metric will only compare the encoding of the entries, i.e., the bitstrings.

Therefore, we are assuming that such a metric preserves the similarity between database entries: similar bitstrings represent similar entries in the database.

The Hamming Distance as Similarity Metric

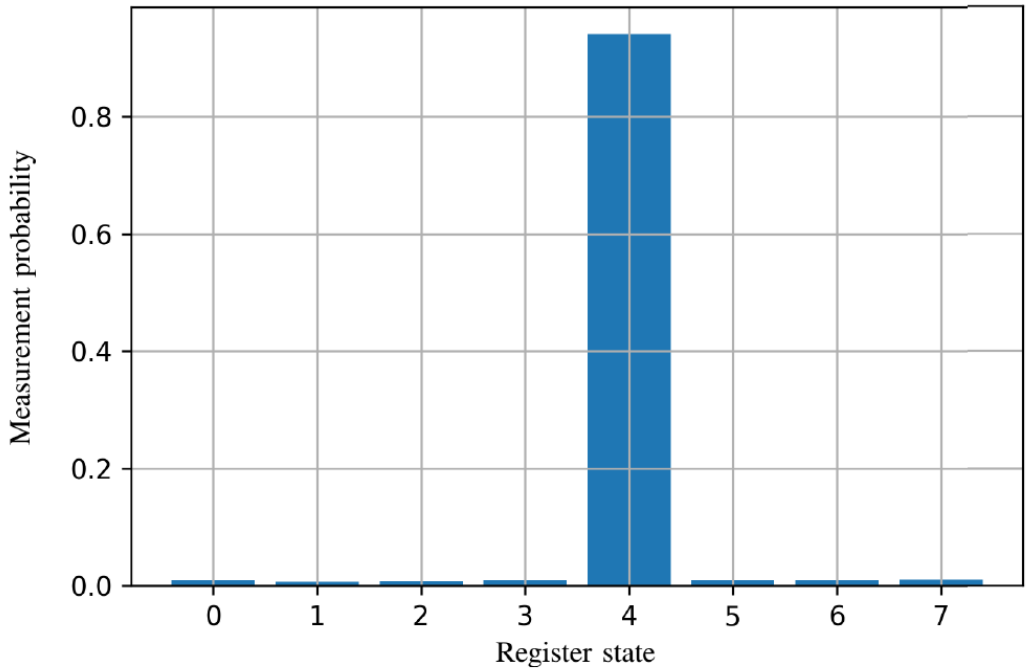
To employ such a similarity metric, the oracle will employ the following **phase tagging** function:

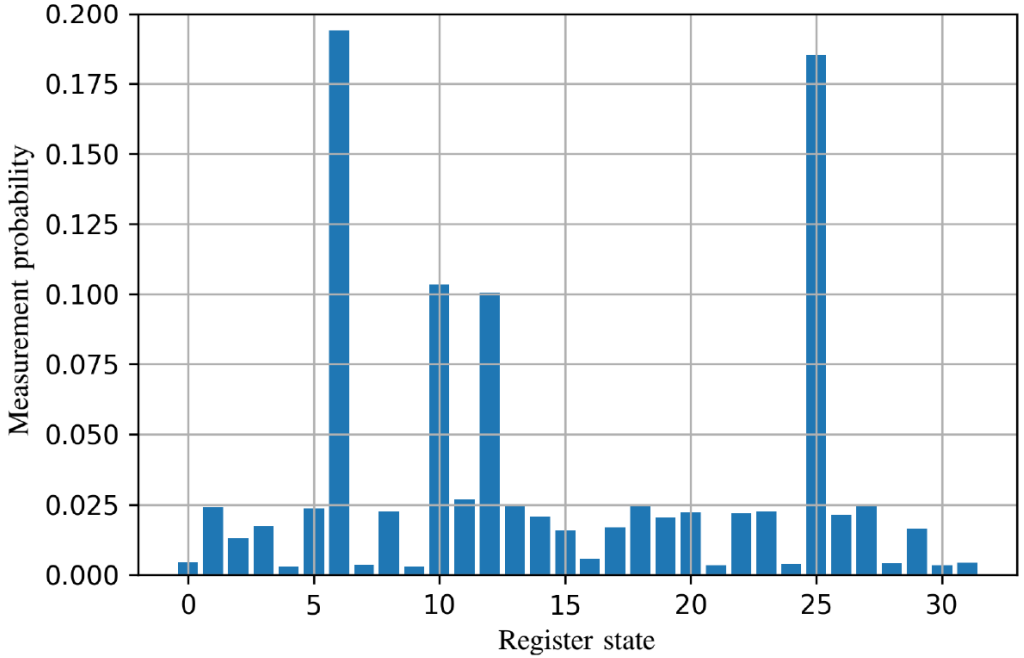
$$T^{\text{sim}}(l(e_q)) = \bigotimes_{j=0}^{k-1} \text{RZ}_i \left(\frac{-(-1)^{l(e_q)_j} 2\pi}{k} \right) = \exp\left(\frac{i\pi}{k} \sum_{j=0}^{k-1} (-1)^{l_j(e_q) \oplus l_j(e)} \right)$$

Notice that, if l(e) = l(e_q), we have $l_i(e_q) \oplus l_i(e) = 0 \quad \forall i < k$, therefore the applied phase is simply π, just like a regular phase tag.
If l(e) = l(e_q) for all but one j-index, the sum evaluates to k – 2. Thus, the phase is $\pi(1 - \frac{2}{k})$.

Results of This Approach

Let's see how the probability amplitude values vary moving from a generic phase tagging function to a similarity-based function.





Higher amplitudes represent a lower Hamming-Distance. Absolute values of the amplitudes are lower, in a kind of «law of conservation of the probability mass».

More Similarity Metrics

We can exploit even more similarity metrics. A generic metric is given in the following form:

$$f : Q \times \mathbb{F}_2^k \rightarrow [0, 1]$$

where Q is the database with the entries to compare, and $\mathbb{F}_2^k$ is the set of the bitstrings of length k (Cartesian product of the Galois field of size 2).

An example is the Dice Coefficient:

$$f : \mathbb{F}_2^k \times \mathbb{F}_2^k \rightarrow [0, 1],$$

$$((x_0, x_1, ..x_k), (y_0, y_1, ..y_k)) \rightarrow \frac{2\sum_{i=0}^{k} x_i y_i}{\sum_{i=0}^{k}(x_i + y_i)}$$

Contrast Functions

As a final remark regarding similarity metrics, it may happen that some metrics produce very close values even if the bitstrings are quite different.

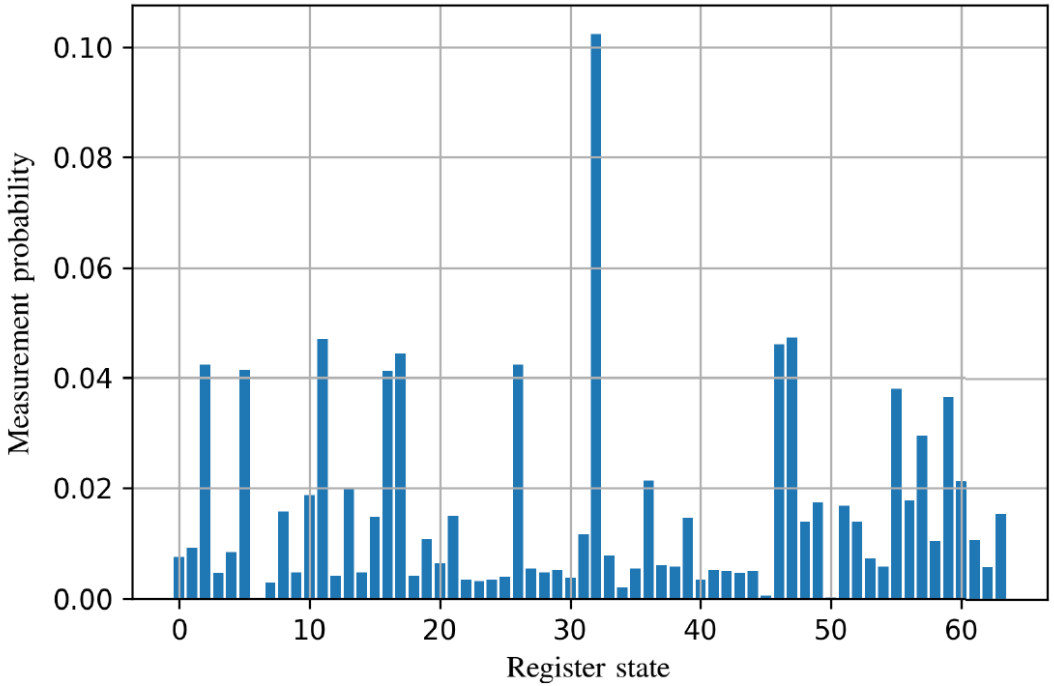
In particular, there may be too many small amplitudes. If we keep into account that amplitudes are proportional to probabilities, too many non-zero values imply reduced probability for the correct output.

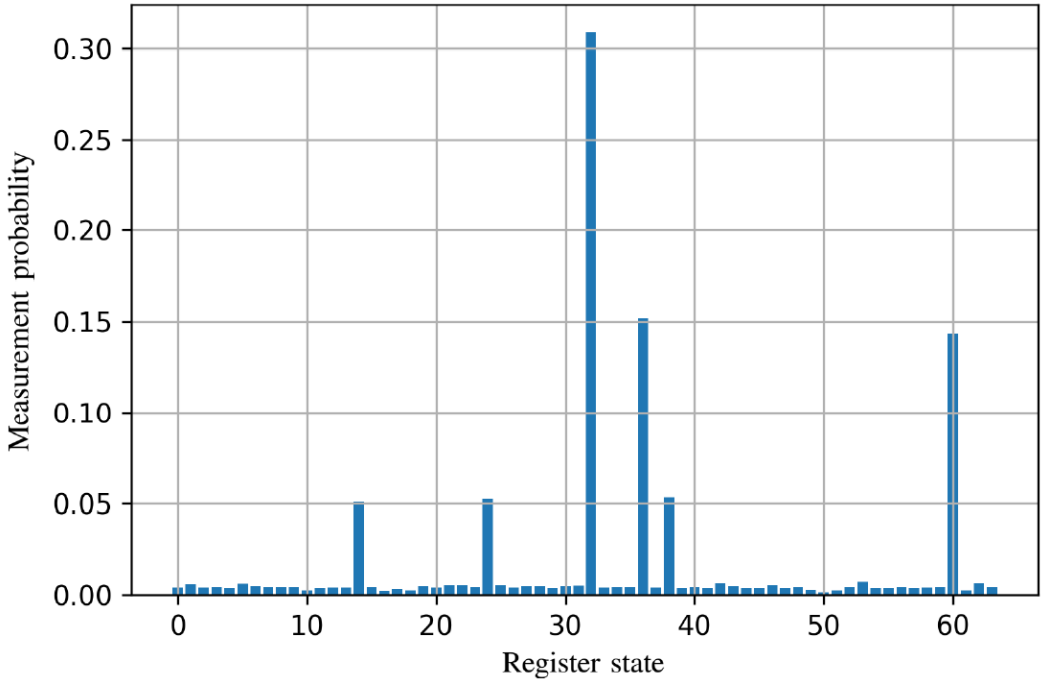
We can use a contrast function to amplify the difference between amplitudes and nullify excessively small values, therefore improving the amplitudes of the correct solutions to Grover's algorithm.

A generic contrast function:

$$\Lambda : [0, 1] \rightarrow [0, 1]$$

Contrast Functions





$$\Lambda(x) = (\exp(30(0.78 - x)) + 1)^{-1}$$

Thank you for your attention!

Quantum Computing
A Practical Perspective

Marco Venere

marco.venere@polimi.it

November 21st, 2024
Politecnico di Milano