Unstructured search

Summary

We'll begin with a description of the problem that Grover's algorithm solves. As usual, we'll let $\Sigma = \{0, 1\}$ denote the binary alphabet throughout this discussion.

Suppose that

$$f : \Sigma^n \to \Sigma$$

is a function from binary strings of length n to bits. We'll assume that we can compute this function efficiently, but otherwise it's arbitrary and we can't rely on it having a special structure or specific implementation that suits our needs.

What Grover's algorithm does is to search for a string $x \in \Sigma^n$ for which $f(x) = 1$. We'll refer to strings like this as *solutions* to the searching problem. If there are multiple solutions, then any one of them is considered to be a correct output, and if there are no solutions, then a correct answer requires that we report that there are no solutions.

This task is described as an *unstructured search* problem because we can't rely on f having any particular structure to make it easy. We're not searching an ordered list, or within some data structure specifically designed to facilitate searching, we're essentially looking for a needle in a haystack.

Intuitively speaking, we might imagine that we have an extremely complicated Boolean circuit that computes f, and we can easily run this circuit on a selected input string if we choose. But because the circuit is so complicated, we have no hope of making sense of the circuit by examining it (beyond having the ability to evaluate it on selected input strings).

One way to perform this searching task classically is to simply iterate through all of the strings $x \in \Sigma^n$, evaluating f on each one to check

whether or not it is a solution. Hereafter, let's write

$$N = 2^n$$

for the sake of convenience. There are N strings in Σ^n, so iterating through all of them requires N evaluations of f. Operating under the assumption that we're limited to evaluating f on chosen inputs, this is the best we can do with a deterministic algorithm if we want to guarantee success. With a probabilistic algorithm, we might hope to save time by randomly choosing input strings to f, but we'll still require $O(N)$ evaluations of f if we want this method to succeed with high probability.

Grover's algorithm solves this search problem with high probability with just $O(\sqrt{N})$ evaluations of f. To be clear, these function evaluations must happen *in superposition*, similar to the query algorithms discussed in the *Quantum query algorithms* lesson, including Deutsch's algorithm, the Deutsch-Jozsa algorithm, and Simon's algorithm. Unlike those algorithms, Grover's algorithm takes an iterative approach: it evaluates f on superpositions of input strings and intersperses these evaluations with other operations that have the effect of creating interference patterns, leading to a solution with high probability (if one exists) after $O(\sqrt{N})$ iterations.

Formal problem statement

We'll formalize the problem that Grover's algorithm solves using the query model of computation. That is, we'll assume that we have access to the function $f : \Sigma^n \to \Sigma$ through a query gate defined in the usual way:

$$U_f\big(|a\rangle|x\rangle\big) = |a \oplus f(x)\rangle|x\rangle$$

for every $x \in \Sigma^n$ and $a \in \Sigma$. This is the action of U_f on standard basis states, and its action in general is determined by linearity.

As was discussed in the *Quantum algorithmic foundations* lesson, if we have a Boolean circuit for computing f, we can transform that Boolean circuit description into a quantum circuit implementing U_f (using some number of workspace qubits that start and end the computation in the $|0\rangle$ state). So, although we're using the query model to formalize the problem that Grover's algorithm solves, it is not limited to this model; we can run Grover's algorithm on any function f for which we have a Boolean circuit.

Here's a precise statement of the problem, which is named *Search* because we're searching for a solution, meaning a string x that causes f to evaluate to 1.

> **Search**
> Input: a function $f : \Sigma^n \to \Sigma$
> Output: a string $x \in \Sigma^n$ satisfying $f(x) = 1$, or "no solution" if no such string x exists

Notice that this is *not* a promise problem — the function f is arbitrary. It will, however, be helpful to consider the following promise variant of the problem, where we're guaranteed that there's exactly one solution. This problem appeared as an example of a promise problem in the *Quantum query algorithms* lesson.

> **Unique search**
> Input: a function of the form $f : \Sigma^n \to \Sigma$
> Promise: there is exactly one string $z \in \Sigma^n$ for which $f(z) = 1$, with $f(x) = 0$ for all strings $x \neq z$
> Output: the string z

Also notice that the *Or* problem mentioned in the same lesson is closely related to *Search*. For that problem, the goal is simply to determine whether or not a solution exists, as opposed to actually finding a solution.

Previous page Next page