A Quantum Computation Primer

Noah Singer

Spring 2018

Contents

1 Introduction **1**

1.1 Notation 3

2 Qubits and gates **4**

2.1 Gates 5

3 Quantum Hadamard analysis **6**

4 Grover's algorithm **9**

5 Simon's problem **10**

6 Quantum Fourier analysis **12**

7 Period finding **14**

8 Shor's algorithm **15**

9 Acknowledgements **16**

1 Introduction

The world we live in is inherently uncertain and random—in other words, quantum. Quantum computing seeks to leverage the quantum mechanical nature of the universe to create algorithms that solve problems significantly more efficiently than is possible using classical computational techniques. Here's a simple and somewhat accurate analogy: If a classical computer tries to find its way through a maze, it is forced to make a guess at every branch and backtrack if it hits a dead end. On the other hand, quantum computers can possibly explore multiple branches of the maze simultaneously. There are no problems that quantum computers can solve that classical computers cannot, but quantum computers do often make it practical to solve certain problems, like integer factorization, which are classically unfeasible due to time constraints. (More on this later.)

So why haven't quantum computers replaced classical computers? Because in physical reality, it's still an open problem how to produce large-scale quantum computers. Recent victories in the field of quantum computing have included building custom-built computers to factor 15 [5] and 21 [1]. In this primer, we'll attempt to build up a working knowledge of quantum computing systems that will help us understand how these victories were achieved.

In classical mechanics, a system exists at all times in a defined *state* which can be *measured*. In quantum mechanics, on the other hand, a system might not exist in a single, classical state. Instead, it can exist in a *superposition* of multiple classical states simultaneously. When the system is observed, the state will correspond to any one of these several superimposed states, each with a certain probability; if the system is measured again, the observed state will not change.

Let's look at a simple example where the "state" that we care about is the position x of a particle. A *wave function*, denoted Ψ, assigns a complex amplitude $\Psi(x)$ to every possible position x. In the standard *Copenhagen interpretation* of quantum mechanics, $\Psi(x)$ corresponds to a *probability* $|\Psi(x)|^2$ of observing the particle at position x. Since the total probabilities must add up to 1, the wave function must satisfy the normality condition

$$\int_{-\infty}^{\infty} |\Psi(x)|^2 dx = 1 \tag{1}$$

As soon as the position of the particle is actually measured and the position x of the particle becomes known exactly, the particle's wave function vanishes except for at x. This process is termed *wave function collapse*; all subsequent measurements will also observe the same value. We didn't know what classical state the system was in, but by observing it, we've forced it to choose one and stick with it. This might appear somewhat confusing—after all, how can a passive observer actually cause such a change in a system? But remember that "passive" observers, in order to do their observing, must interact with the system and exchange information with it via photons or other particles; these interactions are the cause of wave function collapse and other phenomena.

Quantum algorithms are in some ways rather similar to classical probabilistic algorithms. Probabilistic algorithms, instead of always yielding the correct answer (as in traditional deterministic algorithms), are randomized; they are guaranteed to return the correct answer with a probability bounded below[1] by $\varepsilon > 0$.

In probabilistic circuits, instead of being able to completely determine the state of the system, we have a probability distribution of what the system's state could be. For example, given two classical bits, one that has a 30% chance of measuring to be 0 and a 70% chance of measuring to be 1, and another that has a 60% chance of 0 and a 40% chance of 1, if the two bits are combined in an AND gate, we know that we have a 72% chance of measuring the resulting bit to be 0 and a 28% chance of measuring it to be 1; each of these states, a mixture between 0 and 1, is analogous to quantum superposition. If we also combined the bits in an OR gate, we'd have a 21% chance of measuring a 0. But note that these two probabilities are not independent, they are correlated; if we observed the OR gate to output 1, the chance that the AND gate would output 0 is no longer 72%. This is analogous to the quantum phenomenon of *entanglement*.

Quantum computation is essentially an extension of probabilistic classical computation, with one key difference: the probability amplitudes can be complex numbers instead of simply positive reals. *This is the only difference between quantum and classical probabilistic circuits.* Classical circuits "track" positive real amplitudes implicitly; quantum circuits track complex amplitudes implicitly.

The fact that quantum systems track complex amplitudes should make some sense when we think about *wave-particle duality*: the fact that all objects in quantum mechanical systems have particle-like and wave-like properties. In order to describe a wave at any given location, we have to know not only its amplitude but also its phase—and complex numbers capture both of these quantities. To make this more concrete, consider the classic double-slit experiment, where a beam of electrons is passed through two narrow slits onto a phosphorescent film and an interference pattern is observed. Suppose the probability of an electron being at position x when only the first slit is open is $|\psi_A|^2$, and the probability when only the second slit is open is $|\psi_B^2|$. Then what is the probability that, when both slits are open, there is an electron at x? Is it $|\psi_A|^2 + |\psi_B|^2$. No! We also have to account for interference between the beams. This interference term can be shown to equal $2|\psi_1||\psi_2|\cos(\phi_1 - \phi_2)$, where ϕ_1 and ϕ_2 are the *phases* of the beams at x; if we use complex amplitudes, the combined probability amplitude is conventiently $|\psi_A + \psi_B|^2$.

Since classical phenomena are described by real, instead of complex, amplitudes, in order to simulate a quantum system using a classical computer, a classical computer must separately store complex amplitudes for all 2^n possible configurations of quantum bits. This means that it takes $\mathcal{O}(2^n)$ classical bits to accurately simulate n quantum bits. Therefore, quantum algorithms can sometimes achieve exponential speedups over classical algorithms!

We will first examine quantum data in Section 2 and how to transform and analyze that data in Sections 3 and 6. Then, we will demonstrate a classic result, *Grover's algorithm*, which efficiently solves the problem searching through a database with N elements to find an element which makes some predicate f yield 1. Classically, linear search takes time $\Theta(N)$, but Grover's probabilistic algorithm terminates in time $\Theta(\sqrt{N})$. After solving several useful small problems in Sections 5 and

[1]By this, we mean that the algorithm is guaranteed to be correct with probability at least ε. Assuming that we can efficiently check whether the output is correct, this probability can be increased to any probability $p < 1$ by repeating the process a constant number of times—in particular $\log_{1-\varepsilon}(1-p)$.

7, we will conclude demonstrate *Shor's algorithm* for efficient (polynomial-time) integer factorization in Section 8. Shor's algorithm is remarkable because the fastest known classical factoring algorithm, the general field number sieve, requires (roughly) $\mathcal{O}(e^{n/3})$ time to factor an n-bit integer. Many important cryptographic systems, such as RSA, are founded upon on the classical *intractibility* of the integer factorization problem: the (apparent) lack of any algorithm which can factor n-bit integers in polynomial time in n. Thus, Shor's algorithm, if implemented on a large scale, would have gigantic applications for cryptography.

This primer may use a lot of complicated-looking notation, but my goal is for y'all to try and see the big picture of how quantum algorithms work. The exact details don't matter too much. Therefore, don't worry too much if you're not as familiar/comfortable with the notation I'm using. I hope you enjoy!

1.1 Notation

The Hermitian conjugate of A, denoted $A^\dagger$, is formed by taking the complex conjugate of A's entries and then taking the transpose of the resulting matrix.

In quantum mechanis, the notation $|\psi\rangle$ (read: "*ket* ψ") denotes a column vector and $\langle\psi|$ (read: "*bra* ψ") denotes the Hermitian conjugate of $|\psi\rangle$, a row vector.

In linear algebra, the inner product of two complex vectors $x, y \in \mathbb{C}^n$ is $\sum_i \overline{x_i} y_i$ (think about the fact that $|x^2| = \overline{x}x$). In quantum, we write complex inner product of two vectors ψ and ϕ as $\psi^\dagger\phi$ or $\langle\psi|\phi\rangle$.

An $n \times m$ matrix U and a $p \times q$ matrix V can be combined using the *Kronecker tensor product* or *outer product*, denoted $\otimes$, into a $mp \times nq$ matrix

$$U \otimes V = \begin{bmatrix} U_{11}V & \cdots & U_{1n}V \\ \vdots & \ddots & \vdots \\ U_{n1}V & \cdots & U_{nn}V \end{bmatrix} \tag{2}$$

If x and y are binary strings, then xy denotes the binary string "x followed by y"; x^n denotes x repeated n times. We also use the notation $\mathbb{Z}_k$ to denote all natural numbers from 0 to $k-1$, and thus "all numbers from 0 to $2^n - 1$" can be represented as either $\mathbb{Z}_2^n$ or equivalently $\mathbb{Z}_N$. In general, we consider n bits and $N = 2^n$ possible bit-strings of length n. Depending on the context, $\sum_x$ denotes either the sum over all such bit strings of length n ($\mathbb{Z}_2^n$) or simply $\mathbb{Z}_N$. We use $k|n$ to indicate that the integer k "divides" the integer n; that is, that there is an integer j such that $kj = n$.

Throughout this article, we'll play somewhat fast-and-loose with complexities and proofs of convergence due to space constraints. We use the *time complexity* and *circuit complexity* of algorithms interchangeably; technically, the former signifies the amount of time an algorithm takes to compute, whereas the latter signifies the minimum size/depth (in terms of number of simple gates) of a circuit that carries out the algorithm.

Finally, as is the convention in physics, we'll often omit normalization coefficients, since they become unwieldy. You can always add in factors of N as necessary.

2 Qubits and gates

Classical computation centers around observables called *bits* which exists in exactly one of two states, either one/on/yes or zero/off/no. In quantum computing, the standard definition of these values is as the orthonormal basis vectors

$$|0\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, |1\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \tag{3}$$

Classically, a bit is always either $|0\rangle$ or $|1\rangle$. Quantumly, just as with the particle wave function Ψ, a quantum bit—a *qubit*—exists in a superposition of these two basis states:

$$|\psi\rangle = \alpha |0\rangle + \beta |1\rangle = \begin{bmatrix} \alpha \\ \beta \end{bmatrix} \tag{4}$$

where α, β are complex numbers that satisfy a discrete equivalent of the normality condition (1): $\langle\psi|\psi\rangle = |\alpha|^2 + |\beta|^2 = 1$. If we combine two qubits $|\psi_0\rangle = \alpha |0\rangle + \beta |1\rangle$ and $|\psi_1\rangle = \gamma |0\rangle + \delta |1\rangle$ using Equation 2 as $|\psi\rangle = |\psi_0\rangle \otimes |\psi_1\rangle$, we get a 2-qubit system, a superposition

$$(\alpha |0\rangle + \beta |1\rangle) \otimes (\gamma |0\rangle + \delta |1\rangle) = \alpha\gamma |00\rangle + \alpha\delta |01\rangle + \beta\gamma |10\rangle + \beta\delta |11\rangle \tag{5}$$

satisfying a similar condition $|\alpha\gamma|^2 + |\alpha\delta|^2 + |\beta\gamma|^2 + |\beta\delta|^2 = 1$. Again, for example, the probability that when $|\psi\rangle$ is measured, the result is $|10\rangle$, is $|\beta\gamma|^2$.

Qubits can be "implemented" within quantum-level media that have two main "basis" states, for example the spin of a nucleus or electron (up or down) [2]. The main challenge in quantum computing is to design controllable quantum systems which avoid the problem of *decoherence*: that is, the degradation of quantum behavior over time due to thermodynamic/other interactions with the outside environment.

It's also important to note that, while often the most useful, $|0\rangle$ and $|1\rangle$ is not the only possible basis that can represent a qubit $|\psi\rangle$. Any pair of orthogonal vectors will do. Another important basis (as we will see shortly) is

$$|+\rangle = \frac{|0\rangle + |1\rangle}{\sqrt{2}}, |-\rangle = \frac{|0\rangle - |1\rangle}{\sqrt{2}}$$

The *Bell state* is defined as

$$|\psi^+\rangle = \frac{1}{\sqrt{2}} |00\rangle + \frac{1}{\sqrt{2}} |11\rangle$$

or in other words, the state in which there's a 50% chance of observing $|0\rangle$ and $|0\rangle$ and a 50% chance of observing $|1\rangle$ and $|1\rangle$. Any two qubits which together exist in a Bell state are called an

Einstein-Podolsky-Rosen (EPR) pair. EPR pairs are extremely important because they display an essential property of quantum theory. If Alice and Bob possess two qubits that together are in the Bell state and Alice measures her qubit to be $|0\rangle$, then she's entirely certain that Bob will also measure his qubit to be $|0\rangle$, and vice versa. The probability amplitudes of measuring one qubit in the system are not independent of the amplitudes for the other; thus, the two qubits are entangled.

Note that $|\psi^+\rangle$ cannot be factored as the product of two single qubits: continuing from Equation 5, we cannot have $\alpha\gamma = \beta\delta = 1$ but $\alpha\delta = \beta\gamma = 0$. This implies that we cannot decompose the two-qubit system $|\psi^+\rangle$ into two independent single qubits, which is another way of thinking about entanglement.

2.1 Gates

In classical computation, bits can be combined to produce new bits by operators called logic *gates*. Familiar examples include AND, OR, NOT, etc. In many respects, gates are the basic unit of the algorithm: bits are information, and gates transform information. In quantum computation, these gates correspond to operators U which must possess two fundamental properties:

1. For a qubit $|\psi\rangle$, the transformed value $U|\psi\rangle$ must also be a qubit, so the normalization condition $(U|\psi\rangle)^\dagger(U|\psi\rangle) = 1$ is satisfied.

2. The operator must be linear (a matrix).

Thus, since $(U|\psi\rangle)^\dagger(U|\psi\rangle) = \langle\psi| U^\dagger U |\psi\rangle = 1$, we know that $U^\dagger U = I$, so U must be unitary[2]. In fact, any unitary matrix has an equivalent quantum gate!

In general, given a series of gates $U_1, U_2, \ldots, U_k$ that each act on k qubits, we can construct a single gate that acts on a k-qubit system simply by computing $U_1 \otimes U_2 \otimes \cdots \otimes U_k$; an important special case of this is that we can make a single-qubit gate U act on k qubits by tensoring U k times with itself to form $U^{\otimes k}$.

One important thing to note about any quantum gate U is that, since its columns are mutually orthogonal, U has an inverse. Computationally, this means that the U gate must be *reversible*; you cannot lose any information between U's outputs and inputs[3]. Many classical gates are not reversible; for example, AND certainly is not, because it outputs only one bit given two. If we constructed an "augmented AND" gate that, given x_1 and x_2, outputted both x_1 and $x_1 \wedge x_2$, it would still not be reversible because if $x_1 = 0$ then it is impossible to deduce x_2. However, a similar "augmented XOR" would be reversible. This gate, C, an example of a *controlled gate* because its action is essentially controlled by the value of a second qubit, is called the controlled-NOT or CNOT gate (since x_2 is flipped only when $x_1 = |1\rangle$), and is essential for creating EPR pairs.

[2]That is, $U^\dagger = U$. This is the complex analogue of being a symmetric matrix; unitary matrices are necessarily invertible, since their columns are all orthogonal.

[3]One caveat: we are allowed to have junk inputs, called *ancilla* bits, whose values we predefine as $|0\rangle$ or $|1\rangle$.

The single most important gate in quantum computation is H, the *Hadamard gate*. H is defined as

$$H : |0\rangle \mapsto |+\rangle, |1\rangle \mapsto |-\rangle$$

which, due to linearity, is enough to completely characterize it as

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

One important application of H is that, given series of "blank canvas" qubits $|0^n\rangle$, applying $H^{\otimes n}$ creates an equal superposition of all possible states.

Here's an example of a proof using the properties of quantum gates.

Theorem 1 (No-cloning theorem)**.** *There is no quantum gate U that, given a bit $|\phi\rangle$ and $n-1$ ancilla bits $|0^{n-1}\rangle$, outputs $|\phi\rangle \otimes |\phi\rangle \otimes f(|\phi\rangle)$ for any garbage function f. In other words, it is impossible to clone $|\phi\rangle$.*

Proof. Suppose, for contradiction, that U exists. Then $U|0^n\rangle = |00\rangle \otimes f(|0\rangle)$ and $U|10^{n-1}\rangle = |11\rangle \otimes f(|1\rangle)$. Consider

$$\begin{aligned} U(|+\rangle \otimes |0^{n-1}\rangle) &= \frac{1}{\sqrt{2}} U|0^n\rangle + \frac{1}{\sqrt{2}} U|10^{n-1}\rangle \\ &= \frac{1}{\sqrt{2}} |00\rangle \otimes f(|0\rangle) + \frac{1}{\sqrt{2}} |11\rangle \otimes f(|1\rangle) \end{aligned}$$

where the first equality follows from the linearity of U. $U|+\rangle$ essentially yields a Bell state, where measuring yields either $|00\rangle$ and $|11\rangle$ with 50% probability, but if $|+\rangle$ were truly cloned, we would want to measure $|00\rangle, |01\rangle, |10\rangle, |11\rangle$ with equal probability. Thus, U cannot exist. □

The no-cloning theorem, coupled with its counterpart the *no-deletion theorem*, should make some sense, because all quantum operators must be reversible; they cannot create or destroy information, only transform it.

3 Quantum Hadamard analysis

In order to actually solve problems with quantum computers, we must examine different ways of analyzing and extracting useful patterns from quantum information. In particular, we'll examine how functions $g : \mathbb{Z}_2^n \to \mathbb{C}$, which encode probability amplitudes of an n-qubit system, act under various transformations by looking at different *bases* for the vector space of all such functions g. Each basis is a set of functions which are orthonormal under the inner product $\sum_x \overline{\alpha(x)}\beta(x)$. Given a particular basis, every function $g : \mathbb{Z}_2^n \to \mathbb{C}$ can be represented as a linear combination of the basis functions. When we *transform* the quantum system into this new basis, and then measure it, we'll be able to glean different kinds of insights into the problems we're trying to solve as compared to simply measuring the original system.

The most obvious choice for a basis is the *Dirac delta functions* $\{\delta_{\hat{x}}\}_{\hat{x}\in\mathbb{Z}_2^n}$ where

$$\delta_{\hat{x}}(x) = \begin{cases} 1 & x = \hat{x} \\ 0 & x \neq \hat{x} \end{cases} \tag{6}$$

The nice thing about this basis is that, to construct a function g, the coefficient on the basis function $\delta_{\hat{x}}$ is simply $g(\hat{x})$. For example, we can represent the function f that maps 00 to 4, 01 to $-i$, 10 to $2+i$, and 11 to 0 as $f = 4\delta_{00} + (-i)\delta_{01} + (2+i)\delta_{10} + 0\delta_{11}$. In general, we can define any function g as

$$g(x) = \sum_{\hat{x}} g(\hat{x})\delta_{\hat{x}}(x) \tag{7}$$

since $\delta_{\hat{x}}(x)$ is only 1 where $\hat{x} = x$, and that term has value $g(\hat{x}) = g(x)$.

This is the traditional way to think of a function. Now, we will investigate another basis, the *Hadamard* or *Hadamard-Walsh basis*, which has important applications in several major quantum algorithms. Define the operation $\star$ as

$$x \star y = \bigoplus_{k: x_k = 1 \wedge y_k = 1} 1 \tag{8}$$

where $\bigoplus$ means to "take the XOR" over all bits where both x and y are 1. In other words, $x \star y$ represents the *parity*, either 0 (even) or 1 (odd), of the number of bit-locations where both x and y are 1. For example, $110 \star 011 = 1$ because a single location (the second bit) is 1 in both strings; $110 \star 111 = 0$ because two locations (the first and second bits) are 1 in both strings.

Now define the *Walsh functions*

$$\xi_{\hat{x}}(x) = (-1)^{\hat{x}\star x} \tag{9}$$

which are -1 if the parity is odd and 1 if the parity is even. Continuing our example, $\xi_{110}(011) = \xi_{011}(110) = (-1)^1 - 1$ and $\xi_{110}(111) = \xi_{111}(110) = (-1)^0 = 1$. The Walsh functions $\{\xi_{\hat{x}}\}_{\hat{x}\in\mathbb{Z}_2^n}$ form the basis for the Hadamard transform.

Theorem 2. *The Hadamard basis is orthonormal.*

Proof. The inner product of ξ_α and ξ_β, $\sum_x \overline{\xi_\alpha(x)}\xi_\beta(x)$, can be rewritten via Equation 9 as

$$\sum_x (-1)^{(\alpha\star x)+(\beta\star x)} = \sum_x (-1)^{(\alpha\oplus\beta)\star x}.$$

Observe that $\sum_x (-1^{\gamma\star x})$ is 1 exactly when $\gamma = 0^n$ and 0 otherwise. Since the inner product of basis functions ξ_α and ξ_β is 0 exactly when $\alpha + \beta \neq 0^m$—when α and β differ—and 1 when they are the same, the basis is orthonormal. □

This is the essential idea behind the Hadamard transform. We can re-express functions $g(x)$ defined in the standard basis as $\sum_{\hat{x}} g(\hat{x})\sigma_{\hat{x}}(x)$ in the Hadamard basis, using *Hadamard coefficients* $\hat{g}$, as

$$g(x) = \sum_{\hat{x}} \hat{g}(\hat{x})\xi_{\hat{x}}(x) \tag{10}$$

where the coefficients $\hat{g}(\hat{x})$ are each defined as

$$\hat{g}(\hat{x}) = \sum_{x} g(x)\xi_{\hat{x}}(x) \tag{11}$$

The power of the Hadamard transform is thus. Take a quantum system $|\psi\rangle = \sum_x g(x)|x\rangle$, which is a superposition of basis states, weighted by our function g. A Hadamard transform of $|\psi\rangle$ yields a new system

$$|\Psi\rangle = \sum_{\hat{x}} \hat{g}(\hat{x})|\hat{x}\rangle \tag{12}$$

a superposition of basis states weighted by $\hat{g}$, which can then be inverse-transformed to retrieve the original superposition. Intuitively, the Fourier-transformed quantum state is another "representation" of the original state $|\psi\rangle$ that yields different insight into the structure of $|\psi\rangle$.

One useful fact is that $\hat{g}(0^n)$ is the mean of g over all values x, since $\xi_{0^n}(x)$ is always 1.

The quantum Hadamard transform has a simple physical implementation: the Hadamard gate $H^{\otimes n}$! To see why, observe that since $H|0\rangle \mapsto \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ and $H|1\rangle \mapsto \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$, the weight of $|\hat{x}\rangle$ in $H^{\otimes n}|x\rangle$ is $1/2^{n/2}(-1)^{x\star\hat{x}}$; and thus if we apply the Hadamard gate to $\sum_x g(x)|x\rangle$, we'll get $\sum_{\hat{x}}\sum_x 1/2^{n/2}(-1)^{\hat{x}\star x}g(x)|\hat{x}\rangle$, or in other words, $\sum_{\hat{x}} \hat{g}(\hat{x})|\hat{x}\rangle$.

The Hadamard gate is even its own inverse. Many of our important algorithms will consist of essentially taking a state, Hadamard transforming it with a Hadamard gate, modifying the resultant state somehow, and then inverse transforming it with another Hadamard gate.

Now, for the first time, we'll use these techniques to solve a real problem.

Problem 1. *Given a function $f : \mathbb{Z}_2^n \to \mathbb{Z}_2$ that satisfies exactly one of two conditions, determine which it satisfies:*

1. *f always returns* 0
2. *f is balanced* (0 *for exactly half the strings in* $\mathbb{Z}_2^n$)

Classically, linear search solves this problem in $\Theta(N)$ time. Probabilistic solutions are faster, but there exists a quantum algorithm, the *Deutsch-Jozsa algorithm*, that is guaranteed to solve this problem in constant time!

Inside our circuit, we'll represent our function f as a black-box *oracle*[4] quantum gate $O_f^{\pm}$.

[4]We call it an oracle because its inner workings are transparent to the program that's using it: You bow down to the oracle and feed it $|x\rangle$, and it hisses and smokes and spits either $|x\rangle$ or $-|x\rangle$ back at you.

When we talk about the time complexity of the algorithm, we're actually going to analyze the *query complexity*—how many times the algorithm is forced to use $O_f^{\pm}$ to call f. $O_f^{\pm}$ is a quantum gate with the following definition:

$$O_f^{\pm} |x\rangle \mapsto \begin{cases} |x\rangle & f(|x\rangle) = 0 \\ -|x\rangle & f(|x\rangle) = 1 \end{cases} \tag{13}$$

First, we apply $H^{\otimes n}$ to set up an equal superposition of all 2^n states. Then, the $O_f^{\pm}$ gate negates all states for which $f(|x\rangle) = 0$; this will be either all states or exactly half of the states. This new state is

$$|\psi\rangle = \sum_x g(x) |x\rangle$$

where $g(x) = (-1)^{f(|x\rangle)}$. Next, we perform a Hadamard transform to yield a new state

$$|\Psi\rangle = \sum_{\hat{x}} \hat{g}(\hat{x}) |\hat{x}\rangle$$

Next, we measure the system. What's the probability of all qubits measuring $|0^n\rangle$? $g(x)$ is either always -1, if f is always 0, or -1 exactly half the time, if f is 0 half the time.

Now, consider the probability $|\hat{g}(0^n)|^2$ of measuring the state 0^n. Since $\hat{g}(0^n)$ represents the mean of all values of g, if we measure the system and observe $|0^n\rangle$, we know that the mean of g is non-zero, and thus, f is always 0. Conversely, if we do not observe $|0^n\rangle$, we can conclude that the mean of g is zero. We have solved the problem in constant time!

4 Grover's algorithm

Problem 2 (Unstructured search)**.** *Given a function $f : \mathbb{Z}_2^n \to \mathbb{Z}_2$ that has value* 1 *for exactly one $x^* \in \mathbb{Z}_2^n$, compute x^*.*

Classically, the best algorithm to solve the unstructured search problem is *linear search*, which examines each element in a linear order until it finds one satisfying f. This algorithm is optimal classically (including probabilistic algorithms) and takes time $\Theta(N)$.

Grover's algorithm is a quantum algorithm that solves this (very useful) problem in time $\Theta(\sqrt{N})$. Throughout the algorithm, all the amplitudes are actually real, but can be negative. It proceeds as follows:

1. Create an equal superposition of all 2^n states with the Hadamard gate $H^{\otimes n}$
2. Negate the amplitude of the state $|x^*\rangle$ with the oracle $O_f^{\pm}$
3. Reflect all states about the mean amplitude μ with the *Grover diffusion operator D*, which we will construct

During every iteration, since the amplitude of $|x^*\rangle$ becomes negative and therefore farther away from the mean μ, the combination of the negation and reflection operators strictly increases the amplitude of x^* and slightly decreases the amplitude of all non-optimal states. This is an example of the general technique in quantum computing called *amplitude amplification.*

We want the diffusion operator D to replace $|\psi\rangle = \sum_x g(x)\,|x\rangle$ with $D\,|\psi\rangle = \sum_x g'(x)\,|x\rangle$ where $g'(x) = 2\mu - g(x)$, thereby replacing $g(x)$ with $\mu + (\mu - g(x))$. In order to actually construct this magical operator D, let's consider any arbitrary iteration and inspect g via Hadamard analysis:

$$g(x) = \sum_{\hat{x}} \hat{g}(\hat{x})\xi_{\hat{x}}(x) = \hat{g}(0^n)\xi_{0^n}(x) + \sum_{\hat{x}\neq 0^n} \hat{g}(\hat{x})\xi_{\hat{x}}(x) = \mu + \sum_{\hat{x}\neq 0^n} \hat{g}(\hat{x})\xi_{\hat{x}}(x) \tag{14}$$

where the second equality follows from the facts that $\hat{g}(0^n) = \mu$ and $\xi_{0^n}(x) = 1$. g' can be expressed in a strikingly similar form:

$$g'(x) = 2\mu - g(x) = \mu - (g(x) - \mu) = \hat{g}(0^n)\xi_{0^n}(x) - \sum_{\hat{x}\neq 0^n} \hat{g}(\hat{x})\xi_{\hat{x}}(x) \tag{15}$$

In other words, the Hadamard decompositions of g and g' differ only in that the amplitudes of all states $|x\rangle$ where $x \neq 0^n$ are negated. This can be accomplished by a single, simple gate Z_0:

$$Z_0\,|x\rangle \mapsto \begin{cases} |x\rangle & |x\rangle = 0^n \\ -\,|x\rangle & |x\rangle \neq 0^n \end{cases} \tag{16}$$

So, the diffusion operator $D = H^{\otimes n} Z_0 H^{\otimes n}$ Hadamard transforms the state $|\psi\rangle$, negates the amplitudes of all states except $|0^n\rangle$, and then inverts the transform. By repeatedly applying $O_f^{\pm}$ and D for around $\mathcal{O}(\sqrt{N})$ iterations, we get a superposition with a relatively high chance of getting the correct answer. The $\mathcal{O}(\sqrt{N})$ comes from the fact that we may eventually do so many iterations that the amplitudes of the incorrect states become negative, and so the amplitude of the correct state starts decreasing; algebraically, we can prove that this is not the case as long as we do at most $\pi\sqrt{N}/4$ iterations. Thus, Grover's algorithm converges probabilistically with a quadratic speedup over the classical algorithm. Problem solved!

5 Simon's problem

Grover's algorithm achieved a polynomial time speedup over the classical case. But didn't we say that we should be able to do exponentially better in some cases?

Problem 3. *Given a function $f : \mathbb{Z}_2^n \to \mathbb{Z}_2^m$ for which it is guaranteed that there exists some value $s \in \mathbb{Z}_2^n$ such that for any x_1 and x_2, if $f(x_1) = f(x_2)$, then either $x_1 = x_2$, or $x_1 = x_2 \oplus s$, determine s.*

In other words, f is 2-to-1, and we're trying to figure out the "period" s of f. Simon's problem is a kind of "toy" problem whose real utility is yielding insight into solving more complex problems. In

#	**Gate**	$\vert 000\rangle$	$\vert 001\rangle$	$\vert 010\rangle$	$\vert 011\rangle$	$\vert 100\rangle$	$\vert 101\rangle$	$\vert 110\rangle$	$\vert 111\rangle$
—	—	1	0	0	0	0	0	0	0
	$H^{\otimes n}$	0.35	0.35	0.35	0.35	0.35	0.35	0.35	0.35
1	$O_f^{\pm}$	0.35	0.35	0.35	0.35	0.35	0.35	−0.35	0.35
	$H^{\otimes n}$	0.75	−0.25	0.25	0.25	0.25	0.25	−0.25	−0.25
	Z_0	0.75	0.25	−0.25	−0.25	−0.25	−0.25	0.25	0.25
	$H^{\otimes n}$	0.18	0.18	0.18	0.18	0.18	0.18	0.88	0.18
2	$O_f^{\pm}$	0.18	0.18	0.18	0.18	0.18	0.18	−0.88	0.18
	$H^{\otimes n}$	0.12	−0.37	0.37	0.37	0.37	0.37	−0.37	−0.37
	Z_0	0.12	0.37	−0.37	−0.37	−0.37	−0.37	0.37	0.37
	$H^{\otimes n}$	−0.088	−0.088	−0.088	−0.088	−0.088	−0.088	0.97	−0.088
3	$O_f^{\pm}$	−0.088	−0.088	−0.088	−0.088	−0.088	−0.088	−0.97	−0.088
	$H^{\otimes n}$	−0.56	−0.31	0.31	0.31	0.31	0.31	−0.31	−0.31
	Z_0	−0.56	0.31	−0.31	−0.31	−0.31	−0.31	0.31	0.31
	$H^{\otimes n}$	−0.31	−0.31	−0.31	−0.31	−0.31	−0.31	0.57	−0.31

Table 1: A simulated run of Grover's algorithm for $N = 8$ and $x^* = 6 = |110\rangle$. Each cell contains a probability amplitude for a given basis state at a given time in the algorithm. Conveniently, as we have specified Grover's algorithm, all of the amplitudes are in fact real (but possibly negative). First, we prepare a uniform superposition of all 8 possible states. Then, at each iteration, we first negate the amplitude of x^* and then "invert about the mean", which ultimately increases the amplitude of x^* while decreasing the amplitude everywhere else. Note we are only guaranteed convergence with less than than $\pi\sqrt{8}/4 \approx 2.22$ iterations, and indeed, the amplitude of $|110\rangle$ increases from a uniform 0.35 to 0.88 after Round 1 and then to 0.97 after Round 2, but after Round 3, its amplitude decreases to 0.57, since the other amplitudes begin to drag it down. However, if we were to measure the system immediately after Round 2, we'd have a very high—$0.97^2 \approx 94\%$—chance of observing state $|110\rangle$, which is the correct answer!

order to solve Simon's problem, our oracle O_f is defined[5] for $x, y \in \mathbb{Z}_2^n$ as $O_f\,|xy\rangle \mapsto |x(y \oplus f(x))\rangle$.

The algorithm proceeds as follows:

1. Prepare a uniform superposition of all 2^n possible states with $H^{\otimes n}$
2. Add a uniform vector of zeroes $|0^m\rangle$
3. Use the oracle $O_f : |xy\rangle \mapsto |x(y \oplus f(x)\rangle$ to yield a uniform superposition of $|x\rangle \otimes |f(x)\rangle$
4. Measure the $|f(x)\rangle$ qubits
5. Hadamard transform and measure the first n qubits
6. Repeat $\mathcal{O}(n)$ times

Suppose we measure in step (4) the value $|y\rangle = |f(x)\rangle$. Via conditional probability, we have at this point in the algorithm that the state of the first n qubits is a uniform superposition of states $|x\rangle$ and $|x \oplus s\rangle$ such that $f(x) = f(x \oplus s) = y$. Then

[5]We design it in this way so that the gate is reversible; given x and $y \oplus f(x)$ we can recover both x and y.

$$\begin{aligned}
|x\rangle + |x \oplus s\rangle &\overset{H^{\otimes n}}{\mapsto} \sum_{\hat{x}} (-1)^{\hat{x} \star x} |\hat{x}\rangle + \sum_{\hat{x}} (-1)^{\hat{x} \star (x \oplus s)} |\hat{x}\rangle \\
&= \sum_{\hat{x}} \Big((-1)^{\hat{x} \star x} [1 + (-1)^{\hat{x} \star s}] \Big) |\hat{x}\rangle \\
&= \sum_{\hat{x}} |\hat{x}\rangle \begin{cases} 2(-1)^{\hat{x} \star s} & \hat{x} \star s = 0 \\ 0 & \hat{x} \star s = 1 \end{cases}
\end{aligned}$$

Thus, once we measure the remaining n bits, we are guaranteed to measure a configuration $\hat{x}$ such that $\hat{x} \star s = 0$. The essential idea behind Simon's algorithm is that, after repeating these steps algorithm n times, we have $\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_n$. With bounded probability, the $\{\hat{x}_i\}$ are linearly independent, because each $\hat{x}_i$ is selected uniformly randomly from $\mathbb{R}^n$ and must cut down the size of the subspace of $\mathbb{R}^n$ that remains independent of $\hat{x}_1, \ldots, \hat{x}_{i-1}$ by a factor of 2. Thus, we can create a system of n linear equations, $\hat{x}_1 \star s = 0; \hat{x}_2 \star s = 0; \cdots ; \hat{x}_n \star s = 0$ which allows us to solve for s. We have solved Simon's problem in $\mathcal{O}(n)$ time!

6 Quantum Fourier analysis

Our ultimate goal is to understand Shor's algorithm, the polynomial-time algorithm for factoring integers. The important quantum component of Shor's algorithm is the *period-finding algorithm*, which we will examine shortly. In order to do figure out period-finding, we're going to derive a kind of *Fourier analysis* for functions $g : \mathbb{Z}_N \to \mathbb{C}$. Fourier analysis is another kind change-of-basis we can do on the vector space of functions that lets us represent an arbitrary function g as the sum of N periodic functions; it should therefore make some sense that Fourier analysis is useful in the context of period-finding. Roughly, Fourier analysis is the equivalent of Hadamard analysis, except that we're considering periodicity via addition instead of XOR.

Letting ω be the first Nth root of unity (e.g. $e^{2\pi i/N}$), the Fourier basis for the vector space of g functions is the set $\{\sigma_{\hat{x}}\}_{\hat{x} \in \mathbb{Z}_N}$ of periodic complex exponential functions

$$\sigma_{\hat{x}}(x) = \omega^{\hat{x}x} \tag{17}$$

Thus, we re-express qubit systems $|\psi\rangle = \sum_x g(x)\,|x\rangle$ as $|\Psi\rangle = \sum_{\hat{x}} \hat{g}(\hat{x})\,|\hat{x}\rangle$, where

$$\hat{g}(\hat{x}) = \frac{1}{N} \sum_x \overline{\sigma_x(\hat{x})} g(x) = \frac{1}{N} \sum_x \omega^{-\hat{x}x} g(x) \tag{18}$$

Lemma 1. *For $x \in \mathbb{Z}$, $\frac{1}{N} \sum_{i=1}^{N} \omega^{xi}$ is 1 when $x \equiv 0 \pmod{N}$ and 0 otherwise.*

Proof. Case 1: Suppose $x \equiv 0 \pmod{N}$, so $x = kN$. Then every term in the sum is of the form $\omega^{kNi} = 1^{ki} = 1$, and there are N terms, to yield a total of N.

Case 2: Suppose $x \not\equiv 0 \pmod{N}$. Then we have a geometric series

$$\begin{aligned}\frac{1}{N}\sum_{i=1}^{N}\omega^{xi} &= \frac{1}{N}\omega^0\frac{1-\omega^{Nx}}{1-\omega^x}\\ &= \frac{1-\omega^{N^x}}{1-\omega^x}\\ &= \frac{1-1^x}{1-\omega^x}\\ &= 0\end{aligned}$$

□

Theorem 3. *The Fourier basis is orthonormal.*

Proof. Examine the inner product $\sum_i \overline{\sigma_\alpha(x)}\sigma_\beta(x) = \sum_i \omega^{-(\beta-\alpha)x}$, we can use the lemma: The product is 1 exactly when $\alpha \equiv \beta \pmod N$, and 0 otherwise. □

In order to efficiently perform this new Fourier transform, we need to be able to express it in terms of a polynomial number of simple gates. Consider a simple example where $n = 2, N = 4$, and recall that a transform which is correct for all $|x\rangle \in \mathbb{Z}_2^n$ must also be correct for any superposition of these basis states. Then we want the Fourier transform to map

$$|x\rangle \mapsto \omega^0 |00\rangle + \omega^{-x} |01\rangle + \omega^{-2x} |10\rangle + \omega^{-3x} |11\rangle \tag{19}$$

Importantly, this new state $|\hat{x}\rangle$ is *unentangled*, which means it can be decomposed into the product $|\hat{x}_0\rangle \otimes |\hat{x}_1\rangle$ of two single qubits. Specifically, $|\hat{x}_0\rangle = |0\rangle + \omega^{-2x} |1\rangle$ and $|\hat{x}_1\rangle = |1\rangle + \omega^{-x} |1\rangle$. In general, we can decompose the n-qubit Fourier transform $|\hat{x}\rangle$ of $|x\rangle$ as the product of n qubits $|\hat{x}_0\rangle \otimes |\hat{x}_1\rangle \otimes \cdots \otimes |\hat{x}_{n-1}\rangle$, where

$$|\hat{x}_k\rangle = |0\rangle + \omega^{-2^{n-k}x} |1\rangle \tag{20}$$

This is important because it means that the Fourier transform can be accomplished by considering each qubit $|\hat{x}_k\rangle$ separately.

First, write the binary expansion of a basis state $|x\rangle$ as $x_0x_1\cdots x_{n-1}$.

Consider an arbitrary $|\hat{x}_k\rangle = |0\rangle + \omega^{-2^{n-k}x} |1\rangle$ where $k \geq 1$. Remember that, since ω is the first Nth root of unity, $\omega^N = \omega^{2^n} = 1$. Thus, only the last k bits of x are actually relevant to determining the amplitude of $|1\rangle$ in $|\hat{x}_k\rangle$; the jth bit, if 1, contributes a factor of ω^{2^j}. In the case of $j = n-k-1$, $\omega^{2^j x_j} = (-1)^{x_j}$; this is simply the Hadamard gate! The other gates are controlled variants of another important fundamental gate, the *phase shift* gates R_ϕ, which shift the argument of the amplitude of $|1\rangle$ by the angle ϕ and can be represented in matrix form

$$R_\phi = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\phi} \end{bmatrix} \tag{21}$$

Thus, we can produce the qubit $|\hat{x}_k\rangle$ by daisychaining a Hadamard gate on the $n-k-1$th input qubit and then j-controlled $R_{-2^{j-1}/\pi}$ gates for $n-k \leq j < n$, for a total of $\mathcal{O}(n^2)$ gates; classically, the quantum Fourier transform requires $\mathcal{O}(n2^n)$ gates.

Remember, the Fourier transform does not create or destroy information from the original superimposed state; it transforms it in a reversible manner. The Fourier transform of a system of qubits is called its *Fourier spectrum.* Oftentimes quantum algorithms achieve exponential speedups over classical algorithms via *spectral sampling* of a quantum system.

7 Period finding

We first consider the general problem of period finding, which is an extension of Simon's problem to general periodic functions.

Problem 4 (Period finding)**.** *Given a set of finite, discrete "colors" $\mathcal{C} = c_1, \ldots, c_m$, and a function $f : \mathbb{Z}_N \to \mathcal{C}$, and given that there is some $T \mid N$ such that $f(a+Tb) = f(a)$, find T.*

In other words, we're promised that f, which maps integers from 0 to $N-1$ to a set of colors $\mathcal{C}$, is periodic with some period T. Our goal is to find T.

Note that if $N = 2^n$ then there is an efficient classical algorithm: since the period must divide N, then it must be 2^i for some $0 \leq i \leq N$, and so we need only try $n = \log_2 N$ possible periods for a total runtime of $\mathcal{O}(n)$. However, in Section 8, we'll see how our quantum period finding algorithm can easily be extended to cases where the period T does not divide N.

The period finding algorithm proceeds as follows:

1. Prepare a uniform superposition of all N possible states with $H^{\otimes n}$
2. Append a uniform vector of zeroes $|0^m\rangle$, where $m = |\mathcal{C}|$
3. Use the oracle $O_f : |xy\rangle \mapsto |x(f(x) \oplus y)\rangle$ to yield a uniform superposition of $|x\rangle \otimes |f(x)\rangle$
4. Measure the $|f(x)\rangle$ qubits
5. Fourier transform and measure the first n qubits
6. Repeat $\mathcal{O}(1)$ times

Suppose we measure the last m bits to yield a color $c \in \mathcal{C}$. Suppose $i \in \mathbb{Z}_N$ is the first number such that $f(i) = c$. Then clearly the first n qubits conditionally become a uniform superposition of $\sum_{b=0}^{N/T-1} |a+Tb\rangle$. Similarly to our analysis of Simon's algorithm:

$$\sum_{j=0}^{N/T} |a+Tb\rangle \overset{\mathcal{F}_n}{\mapsto} \sum_{\hat{x}} \left(\sum_{b=0}^{N/T-1} \omega^{-\hat{x}(a+Tb)} \right) |\hat{x}\rangle \tag{22}$$

The above lemma implies that the amplitude of $|\hat{x}\rangle$ is only nonzero when $T\hat{x} \equiv 0 \pmod{N}$. Thus, we have discovered an x which is an integer multiple of the *frequency* $f = \frac{N}{T}$. For those of you who are knowledgeable about Fourier analysis, this shouldn't be surprising—we Fourier decomposed a periodic signal and measured a peak in the frequency domain, which corresponds to the fundamental frequency f or one of its overtones.

Suppose we repeat this process and now have measured af and bf for some (possibly) distinct integers a and b. Since we are sampling from a uniform distribution (note that all non-zero n-tuples of qubits in Equation 22 have the same weight), there is a high chance that a and b are both nonzero. The probability that a and b share a prime factor is low; in fact, since a and b are uniformly distributed, it is bounded above by $1/p^2$ for any particular prime p. Since $\sum_{p \text{ is prime}} 1/p^2$ is asymptotic to $\pi^2/6$, we have a high probability that a and b share no common factors. In this case, we can simply compute $\gcd(af, bf) = f$, and we have found the period in $\mathcal{O}(n)$ time!

8 Shor's algorithm

Shor's algorithm is essentially a classical reduction of the integer factorization problem to a modified form[6] of the period-finding problem in which the period T does not necessarily divide N.

Problem 5. *Given some $N \in \mathbb{N}$, find $q \neq 1, N$ such that $q|N$.*

The first step of Shor's algorithm is to reduce the problem of factorization to the problem of finding a nontrivial square room modulo N, that is, a number r such that $r^2 \equiv 1 \pmod{N}$ such that $r \not\equiv \pm 1 \pmod{N}$. If we find such an r, then we know that either $r - 1 \equiv 0 \pmod{N}$ or $r + 1 \equiv 0 \pmod{N}$, and neither is 0 or N, and so we have discovered a nontrivial factor of N.

Suppose we have some A coprime[7] with N. The *order* of A is the least non-zero integer s such that $A^s \equiv 1 \pmod{N}$. If we're lucky, we'll find that s is even, and that $A^{s/2} \not\equiv -1 \pmod{N}$ (we already know it cannot be 1 since s is minimal). In this case, we have some r such that $r^2 \equiv 1 \pmod{N}$: simply $A^{s/2}$. Turns out, though, we don't actually have to be *that* lucky—if we select A uniformly at random, these conditions are satisfied with some constant probability due to Fermat's little theorem (in the case of prime modulus) and then the Chinese remainder theorem (in the general case)[8] [4].

We have reduced the factoring problem to the problem of, given A and N, finding the least s such that $A^s \equiv 1 \pmod{N}$. We'll call this problem *order-finding*. This problem is identical to period-finding for the function $f : \mathbb{Z}_Q \to \mathbb{Z}^N$ where $f(x) = A^x \pmod{N}$, except for one small caveat: since we don't yet know the factorization of N, we can't be sure what value of Q to pick such that $s|Q$; in other words, the period might not exactly divide the size of the list we're testing.

[6]Classical algorithms can actually solve the period-finding problem for $T|2^n$ in $\mathcal{O}(n^2)$ time, simply by checking all n possible periods. However, classical algorithms take exponential time without the divisibility guarantee, whereas the algorithm we presented above can easily be extended to this case.

[7]A number that shares no prime factors with N; in other words, $\gcd(A, N) = 1$.

[8]There's actually another case to consider: the case where A is an odd power of a prime, in which case no nontrivial square root exists. However, numbers of this form are easy to factor by other means, specifically binary search on the exponent.

Roughly, the algorithm proceeds as follows: it picks a Q much larger than N, but still polynomial in 2^n. Recall that our goal is to find s. When we sample the Fourier spectrum, by a similar argument to the previous section's, we have a high probability of measuring $x = \lceil \frac{kQ}{s} \rceil$ for some $k \in \mathbb{Z}_s$ (as opposed to being able to measure kQ/s directly). Now consider the fraction $\dfrac{x}{N}$. Using a common classical number theory trick—the *continued fraction expansion* of a fraction/real number y as

$$a_0, a_0 + \frac{1}{a_1}, a_0 + \frac{1}{a_1 + \frac{1}{a_2}}, a_0 + \frac{1}{a_1 + \frac{1}{a_2 + \frac{1}{a_3}}}, \dots \tag{23}$$

—we can expand $\dfrac{x}{Q}$ via the Euclidean algorithm into a series of fractions which approximate it. Each resultant fraction must satisfy two conditions: the denominator is less than N, and the whole fraction is asymptotic to $\dfrac{x}{Q}$. One of these fractions is very likely to be the fraction, k/s, that we seek, or a factor of it. Repeat a polynomial number of times, and et voilà, we have found s, and therefore factored N!

9 Acknowledgements

I owe a great debt to Ryan O'Donnell, upon whose course [3], CMU 15-859BB, this primer is based.

References

[1] E. Martín-López et al. "Experimental realization of Shor's quantum factoring algorithm using qubit recycling". In: *Nature Photonics* 6 (2012), pp. 773–776. ISSN: 17494893. DOI: 10.1038/nphoton.2012.259. URL: https://www.nature.com/articles/nphoton.2012.259 (cit. on p. 1).

[2] J.J.L. Morton. In: *Nature* 455 (2008), pp. 1085–1088. ISSN: 14764687. DOI: doi:10.1038/nature07295. URL: https://www.nature.com/articles/nature07295 (cit. on p. 4).

[3] R. O'Donnell and J. Wright. *15-859BB: Quantum Computation and Information 2015*. 2015. URL: https://www.cs.cmu.edu/~odonnell/quantum15/ (cit. on p. 16).

[4] U. V. *CS 294-4 Quantum Computing; Lecture 9: Shor's Factoring Algorithm*. 2004. URL: https://people.eecs.berkeley.edu/~vazirani/f04quantum/notes/lec9.pdf (cit. on p. 15).

[5] L. M. K. Vandersypen et al. "Experimental realization of Shor's quantum factoring algorithm using nuclear magnetic resonance". In: *Nature* 414 (2001), pp. 883–887. ISSN: 14764687. DOI: 10.1038/414883a. URL: https://www.nature.com/articles/414883a (cit. on p. 1).