August 2022

WORKING PAPER SERIES

2022-OPS-01

Quantum Computing in Operations Research

Stefan Creemers
IESEG School of Management, Lille, France
KU Leuven, ORSTAT, Leuven, Belgium

Perez Armas Luis Fernando
IESEG School of Management, Lille, France

IESEG Working Paper Series 2022-OPS-01

Quantum Computing in Operations Research

Stefan Creemers[*]
IESEG School of Management, Lille, France, s.creemers@ieseg.fr
KU Leuven, ORSTAT, Leuven, Belgium, stefan.creemers@kuleuven.be

Perez Armas Luis Fernando
IESEG School of Management, Lille, France, l.perezarmas@ieseg.fr

Quantum computing has sparked a tremendous interest from governments, academics, and the private sector alike. According to a 2021 McKinsey report, governments have announced to invest almost $30 billion to develop quantum technologies. Companies such as IBM, Google, Amazon, and Microsoft are also investing heavily, and have already launched commercial quantum-computing services. Research on quantum computing is also on the rise with hundreds of publications in Nature, Science, and PNAS. Despite this enormous interest, quantum computing has received little or no attention in the OR community. This is somewhat surprising given the potential that has been ascribed to quantum computers to solve Operations Research (OR) problems. To investigate the potential of quantum computing from an OR perspective, we discuss the most important quantum algorithms, and use them to effectively solve the knapsack problem for the very first time. We verify our results using Qiskit (IBM's software development kit for quantum computing), and make available templates that allow to solve other OR problems. In addition, we highlight a number of important limitations and drawbacks of quantum computing (when compared to classical computing), and conclude that quantum computing indeed shows promise, albeit not for every application.

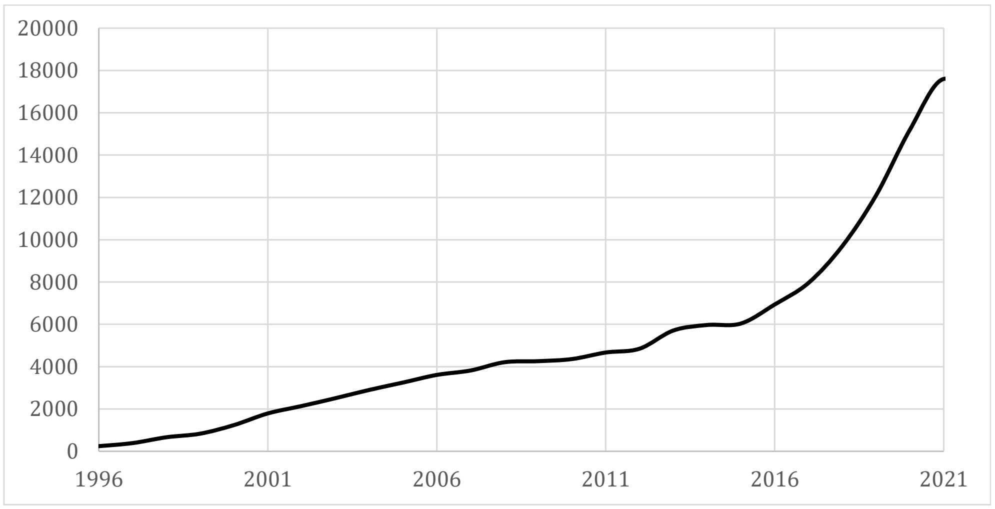
Keywords: Quantum; computing; algorithm; knapsack.

1. Introduction

The idea of quantum computing was first launched in 1980, when Benioff described a quantum mechanical model of a Turing machine. A few years later, Feynman (1982) proposes the idea of a universal quantum simulator. Building on the work of Benioff and Feynman, Deutsch (1985) describes the first universal quantum computer that is able to efficiently simulate any other quantum computer. In addition, Deutsch proposes the Deutsch algorithm, the first quantum algorithm that has a proven speedup when compared to a classical algorithm. Later, Deutsch and Josza (1992) extend this algorithm, and achieve an exponential speedup. Deutsch and Josza, however, consider a problem (the Deutsch-Josza problem) that is purely theoretical and has no practical use. It isn't until 1994 that quantum computing really takes off. In 1994, Shor proposes a quantum algorithm

[*]Corresponding author

Figure 1. Number of English documents that have been published according to Google Scholar when conducting a search using the keyword “quantum computing”.



to find the prime factors of large integers in polynomial time. In theory, Shor’s algorithm can be used to break many of the cryptography schemes in use today. Not surprisingly, the publication of Shor’s algorithm has sparked tremendous interest for quantum computing. A few year later, Grover (1996) published a quantum search algorithm that achieves a quadratic speedup when compared to a classical algorithm. Arguably, Grover’s algorithm is the most important algorithm in quantum computing today. It can be used to solve a variety of problems, and is the corner stone of many other quantum algorithms. Even though many other quantum algorithms have been published since 1997 (e.g., the Quantum Algorithm Zoo lists 62 quantum algorithms), there have been no other major breakthroughs in quantum computing. Despite the lack of recent breakthroughs, a recent McKinsey report (2021) shows that, since 2010, quantum computing has attracted \$3.3 billion in investments from private companies (of which \$1.7 billion in 2021 alone), and that governments have announced to spend almost \$30 billion on the development of quantum technologies. In addition, investments in quantum computing are expected to further increase in the upcoming years.

This increasing trend can also be seen in the number of published papers that deal with quantum computing. For instance, Figure 1 shows the number of English documents that have been published since 1996 according to Google Scholar when conducting a search using the keyword “quantum computing”. Most papers on quantum computing have been published in physics journals, however, also Nature, Science, and PNAS are popular outlets for this topic (with 794, 410, and 115 published papers that mention “quantum computing”, respectively).

Notwithstanding the tremendous interest in quantum computing, there has been little to no attention to this topic from the OR community. This is somewhat surprising given the potential that has been ascribed to quantum computers to solve OR problems. In this paper, we open up the field of quantum computing to the OR community, illustrate the most important quantum algorithms, and show how they can be used to solve the knapsack problem. We highlight challenges that are typically overlooked in the current literature on quantum computing, and verify our calculations using Qiskit; IBM's open-source software development kit that allows to simulate or operate a universal quantum computer. Our code is made available as supplementary material, and includes templates to solve the knapsack problem, the graph-coloring problem, and the resource-constrained project scheduling problem. Using these templates, other OR problems can be solved as well.

Following our implementation of the knapsack problem, we identify a number of important drawbacks that limit the potential of quantum computing to solve OR problems. We discuss how these drawbacks have been addressed in the literature, and show that the proposed solutions also have limitations. we conclude that quantum computing indeed shows promise, albeit not for every application.

In what follows, in Section 2, we introduce the fundamentals of quantum computing, and the notation that will be used throughout the rest of the paper. Next, in Section 3, we define the gates and circuits that are required to establish universal quantum computing. In Section 4, we introduce the Deutsch algorithm and its extension to the Deutsch-Josza algorithm. The Deutsch-Josza algorithm serves as an important building block of Grover's algorithm, that is discussed in Section 5. In Section 6, we use Grover's algorithm to solve the knapsack problem. Section 7 discusses the (current) limitations of quantum computing for solving OR problems, and Section 8 concludes this paper.

2. Fundamentals of Quantum Computing

In 1939, Dirac introduced the "bra-ket" notation to facilitate the study of quantum systems. In this notation, a "ket" $|v\rangle$ denotes a column vector v defined in a complex vector space V, a "bra" $\langle f|$ denotes a row vector f that acts as a linear map on V, and a "bra-ket" $\langle f|v\rangle$ represents the inner product of $\langle f|$ and $|v\rangle$. In what follows, we adopt bra-ket notation.

In quantum computing, information is stored in so-called "qubits". A qubit can be seen as a two-state quantum system that has basis states $|0\rangle = (1,0)^\mathsf{T}$ and $|1\rangle = (0,1)^\mathsf{T}$. These basis states are the quantum equivalents of states 0 and 1 of a classical bit. In contrast to a classical bit, however, a qubit may also be in a superposition of both basis states (i.e., it is in both states at the same time). In superposition, the state of the quantum system is described by a so-called "wave function". In case of a single qubit, the wave function is a linear combination of basis states $|0\rangle$ and $|1\rangle$:

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle\,, \tag{1}$$

where α and β are complex numbers that are also known as the "probability amplitudes" of wave function $|\psi\rangle$. By observing (i.e., measuring) a qubit in superposition, the state of the quantum system is said to collapse into one of the basis states. The probability that a qubit collapses into basis state $|0\rangle$ or $|1\rangle$ is proportional to the square of probability amplitutes α and β, respectively. This result is also known as the Born rule (1926). Given that α and β are complex numbers, we have:

$$|\alpha|^2 + |\beta|^2 = 1. \tag{2}$$

In systems with two (or more) qubits, the basis states can be obtained as the Kronecker product of the basis states of each individual qubit. For instance, if we consider two qubits, a first basis state is given by:

$$|00\rangle = |0\rangle \otimes |0\rangle = (1,0)^{\mathsf{T}} \otimes (1,0)^{\mathsf{T}} = (1,0,0,0)^{\mathsf{T}}.$$

The other basis states are $|01\rangle = (0,1,0,0)^{\mathsf{T}}$, $|10\rangle = (0,0,1,0)^{\mathsf{T}}$, and $|11\rangle = (0,0,0,1)^{\mathsf{T}}$. If both qubits are in superposition, we get:

$$|\boldsymbol{\Psi}\rangle = |\psi_1\rangle \otimes |\psi_2\rangle = (\alpha_1 |0\rangle + \beta_1 |1\rangle) \otimes (\alpha_2 |0\rangle + \beta_2 |1\rangle) = \alpha_1\alpha_2 |00\rangle + \alpha_1\beta_2 |01\rangle + \beta_1\alpha_2 |10\rangle + \beta_1\beta_2 |11\rangle\,,$$

where $|\boldsymbol{\Psi}\rangle$ is used to denote the state of a set of qubits.

3. Universal Quantum Computing

Similar to universal computing, quantum computing uses logic gates to perform operations. In contrast to the gates of classical computing, however, quantum gates need to be reversible (i.e., given the output of an operation, we should be able to determine the input that produced that output). In what follows, we introduce the X, CX, and Toffoli quantum gates, and show how these gates can be used to create the quantum counterpart of the AND, OR, XOR, and NOT gates of classical computing.

The X gate is the quantum equivalent of the classical NOT gate, and transforms input state $|0\rangle$ into output state $|1\rangle$ (and vice versa). Formally, X is a two-by-two matrix that operates as follows:

$$\mathrm{X}|0\rangle = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix} = |1\rangle\,,\;\; \mathrm{X}|1\rangle = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix} = |0\rangle\,.$$

The CX (or controlled-X) gate has two input qubits: a target qubit on which an X operation may be performed, and a control qubit that determines whether or not the X operation is performed on the target qubit. If the control qubit is in basis state $|1\rangle$, the X operation is performed on the target qubit. Otherwise, if the control qubit is in basis state $|0\rangle$, the target qubit remains unchanged. Formally, CX is a four-by-four matrix that operates as follows:

$$\mathrm{CX}|00\rangle = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} = |00\rangle\,.$$

One can verify that $\mathrm{CX}|10\rangle = |11\rangle$, $\mathrm{CX}|01\rangle = |01\rangle$, and $\mathrm{CX}|11\rangle = |10\rangle$.

The Toffoli gate is a controlled-CX (CCX) gate that has two control qubits. Only if both control qubits are in basis state $|1\rangle$, the X operation is performed on the target qubit. Formally, CCX is an eight-by-eight matrix that operates as follows:

$$\text{CCX}\,|000\rangle = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} = |000\rangle\,.$$

One can verify that $\text{CCX}\,|100\rangle = |100\rangle$, $\text{CCX}\,|010\rangle = |010\rangle$, $\text{CCX}\,|110\rangle = |111\rangle$, $\text{CCX}\,|001\rangle = |001\rangle$, $\text{CCX}\,|101\rangle = |101\rangle$, $\text{CCX}\,|011\rangle = |011\rangle$, and $\text{CCX}\,|111\rangle = |110\rangle$.

We can use the X, CX, and CCX gates to create the quantum equivalents of the classical AND, OR, XOR, and NOT gates: the quantum equivalent of the AND gate corresponds to a CCX gate with target qubit initialized as $|0\rangle$, the quantum equivalent of the OR gate is a series of a CX, X, CCX, and X gate with target qubit initialized as $|0\rangle$, the quantum equivalent of the XOR gate is a series of two CX gates with target qubit initialized as $|0\rangle$, and the quantum equivalent of the NOT gate is simply the X gate. All aforementioned gates are illustrated in Figure 2. Note that a series of gates is called a "circuit", and that the quantum equivalent of an AND, OR, and NOT gate suffice to establish universal quantum computing.

4. The Deutsch algorithm

The Deutsch algorithm was first published in 1985, has been generalized by Deutsch and Josza in 1992, and has further been improved by Cleve et al. (1998). Even though the Deutsch algorithm solves a problem that is of no practical use, it is of particular interest because (1) it uses the "phase kickback" trick that is exploited in many quantum algorithms, and (2) it is one of the first quantum algorithms for which it has been shown that it can solve a problem more efficiently than a classical algorithm.

The Deutsch algorithm solves the following problem: we are given a function f that maps a binary input to a binary output in a way that is unknown to us. Even though function f acts as a black box, we are told that f is either "constant" or "balanced"; f is said to be constant if it always produces the same output, and f is balance if it produces a zero for half of the inputs and a 1 for the other half of the inputs. For a binary input, this implies that a balanced function produces a

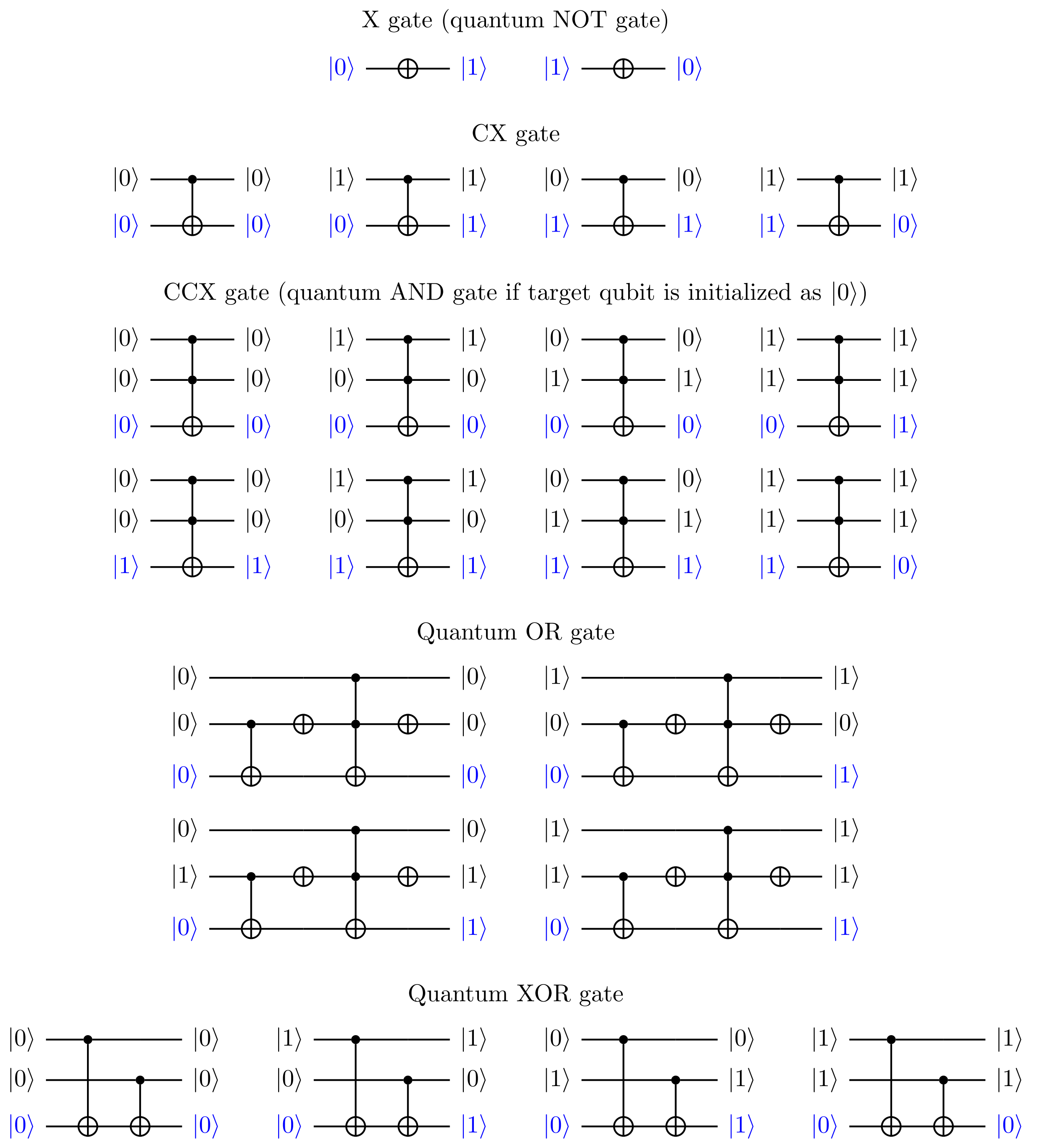

X gate (quantum NOT gate)
CX gate
CCX gate (quantum AND gate if target qubit is initialized as |0⟩)
Quantum OR gate
Quantum XOR gate

Figure 2. Circuits of the X, CX, and CCX quantum gates, as well as the quantum equivalents of the classical AND, OR, XOR, and NOT gates. The target qubit is indicated in blue.

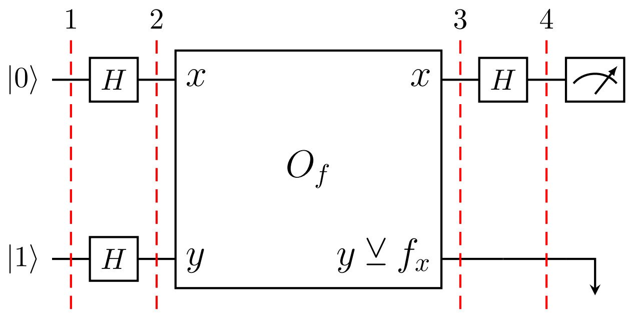
different output for each input. As such, f is one of the following four functions:

$$\begin{aligned}
f_0 = 0 \text{ and } f_1 = 0 \ (f \text{ is constant}),\\
f_0 = 1 \text{ and } f_1 = 0 \ (f \text{ is balanced}),\\
f_0 = 0 \text{ and } f_1 = 1 \ (f \text{ is balanced}),\\
f_0 = 1 \text{ and } f_1 = 1 \ (f \text{ is constant}),
\end{aligned}$$

where f_x is used to denote $f(x)$. To solve the Deutsch problem, it suffices to determine whether f is constant or balanced. The traditional way to solve this problem would be to evaluate both f_0 and f_1, and to observe whether the outputs are the same. This solution, however, requires two calls to function f. Using the Deutsch algorithm, we only require a single call to function f.

The circuit of the Deutsch algorithm is presented in Figure 3.

Figure 3. Circuit of the Deutsch algorithm



In this circuit, the Hadamard gate (represented by the letter H) plays a pivotal role. The Hadamard gate is one of the most important quantum gates as it is used to put qubits in superposition. In case of a single qubit, it operates as follows:

$$\mathrm{H}\,|0\rangle = \frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}\begin{pmatrix}1\\0\end{pmatrix} = \frac{|0\rangle + |1\rangle}{\sqrt{2}} = |+\rangle\,,\ \ \mathrm{H}\,|1\rangle = \frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}\begin{pmatrix}0\\1\end{pmatrix} = \frac{|0\rangle - |1\rangle}{\sqrt{2}} = |-\rangle\,.$$

If a qubit, initialized as $|0\rangle$ or $|1\rangle$, is passed through a Hadamard gate, it is said to be in superposition, and has a 50 percent probability to collapse into either basis state when measured. In general, when applied to a set of n qubits, the Hadamard gate is defined as the n^{th} tensor power of H (denoted $\mathrm{H}^{\otimes n}$), and the probability to measure any of the 2^n basis states equals 2^{-n}. Note that $\mathrm{H}^{\otimes n}$ is an involutory matrix (i.e., a square matrix that is its own inverse). As a result, $\mathrm{H}\,|+\rangle = |0\rangle$ and $\mathrm{H}\,|-\rangle = |1\rangle$.

The Deutsch algorithm can be broken down in four steps (indicated by the red dashed lines in the circuit). In a first step, we initialize two qubits, and obtain quantum state $|\mathbf{\Psi}_1\rangle = |01\rangle$. Next, in a second step, both qubits are put in superposition using a Hadamard gate:

$$|\mathbf{\Psi}_2\rangle = H^{\otimes 2}|01\rangle = \frac{1}{2}\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix}\begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} = \frac{1}{2}\begin{pmatrix} 1 \\ -1 \\ 1 \\ -1 \end{pmatrix} = \frac{1}{2}(|00\rangle - |01\rangle + |10\rangle - |11\rangle) = |{+}{-}\rangle .$$

In step 3, we pass both qubits through an oracle O_f that, unlike us, has complete knowledge of function f. Oracle O_f uses its knowledge of f to map state $|xy\rangle$ to $|x(y \veebar f_x)\rangle$, where $\veebar$ represents the XOR operation, and where x and y represent the first and second qubit, respectively. It is important to note that oracle O_f only makes a single call to function f; only the first qubit is evaluated using f. After applying oracle O_f, we get:

$$|\mathbf{\Psi}_3\rangle = \frac{1}{2}\left(|0\rangle \otimes (|(0 \veebar f_0)\rangle - |(1 \veebar f_0)\rangle) + |1\rangle \otimes (|(0 \veebar f_1)\rangle - |(1 \veebar f_1)\rangle)\right).$$

For $x \in \{0, 1\}$, one can verify that $|x\rangle \otimes (|0 \veebar f_x\rangle - |1 \veebar f_x\rangle) = (-1)^{f_x}|x\rangle \otimes (|0\rangle - |1\rangle)$, and hence, the sign (or phase) of the second qubit is "kicked back" to the first qubit if $f_x = 1$ (i.e., the phase kickback trick has been applied). As a result, $|\mathbf{\Psi}_3\rangle$ may also be defined as:

$$|\mathbf{\Psi}_3\rangle = \frac{1}{2}\Big((-1)^{f_0}|0\rangle \otimes (|0\rangle - |1\rangle) + (-1)^{f_1}|1\rangle \otimes (|0\rangle - |1\rangle)\Big).$$

Quantum state $|\mathbf{\Psi}_3\rangle$ can further be simplified depending on whether f is constant or balanced. First, assume that f is constant. In this case, $f_0 = f_1$, and there are two options. Either $f_0 = 0$, and $|\mathbf{\Psi}_3\rangle$ is given by:

$$|\mathbf{\Psi}_3\rangle = \frac{1}{2}(|00\rangle - |01\rangle + |10\rangle - |11\rangle) = \frac{|0\rangle + |1\rangle}{\sqrt{2}} \otimes \frac{|0\rangle - |1\rangle}{\sqrt{2}} = |{+}{-}\rangle ,$$

or $f_0 = 1$, and we obtain:

$$|\mathbf{\Psi}_3\rangle = \frac{1}{2}(-|00\rangle + |01\rangle - |10\rangle + |11\rangle) = -\frac{|0\rangle + |1\rangle}{\sqrt{2}} \otimes \frac{|0\rangle - |1\rangle}{\sqrt{2}} = -|{+}{-}\rangle .$$

If, on the other hand, f is balanced, $f_0 \neq f_1$, and there are again two options. Either $f_0 = 0$, and $|\mathbf{\Psi}_3\rangle$ is given by:

$$|\mathbf{\Psi}_3\rangle = \frac{1}{2}(|00\rangle - |01\rangle - |10\rangle + |11\rangle) = \frac{|0\rangle - |1\rangle}{\sqrt{2}} \otimes \frac{|0\rangle - |1\rangle}{\sqrt{2}} = |{-}{-}\rangle ,$$

or $f_0 = 1$, and we obtain:

$$|\mathbf{\Psi}_3\rangle = \frac{1}{2}(-|00\rangle + |01\rangle + |10\rangle - |11\rangle) = -\frac{|0\rangle - |1\rangle}{\sqrt{2}} \otimes \frac{|0\rangle - |1\rangle}{\sqrt{2}} = -|{-}{-}\rangle .$$

Note that $-|{+}{-}\rangle$ and $|{+}{-}\rangle$ are said be equivalent; the "global phase" of the quantum system has no impact on the probability that it collapses into one of the basis states. The same logic applies to $-|--\rangle$ and $|--\rangle$. At the end of step 3, we can dispose of the second qubit (it is no longer needed), and observe that the first qubit is in state $|+\rangle$ if f is constant, and in state $|-\rangle$ if f is balanced. Therefore, if the first qubit is passed through a second Hadamard in the last step, we measure $|\mathbf{\Psi}_4\rangle = H|+\rangle = |0\rangle$ if f is constant, and $|\mathbf{\Psi}_4\rangle = H|-\rangle = |1\rangle$ if f is balanced. With only a single call to function f, the Deutsch algorithm is able to determine whether f is constant or balanced. This result was further generalized by Deutsch and Josza, who consider a function f that has n binary inputs (rather than only a single binary input, as was the case in the Deutsch algorithm).

5. Grover's algorithm

The goal of Grover's algorithm is to find a single target entry in an unstructured database that has 2^n entries. Similar to Deutsch and Josza, Grover considers a function f that has n binary inputs that are modeled using n qubits. The resulting quantum system has 2^n basis states that each correspond to an entry of the database. In contrast to Deutsch and Josza, however, f is not constant or balanced, but returns 1 if the target entry is given as an input, and 0 otherwise. Table 1 provides an example with three binary inputs, resulting in a quantum system that has eight basis states. In this example, function f outputs 1 only for target entry 101 (represented by basis state $|101\rangle$; indicated in blue). Throughout the remainder of this section, this example will be used to illustrate the different steps of Grover's algorithm.

Table 1. Example database with eight entries that are represented by the basis states of a quantum system that has three qubits. The target entry is printed in blue.

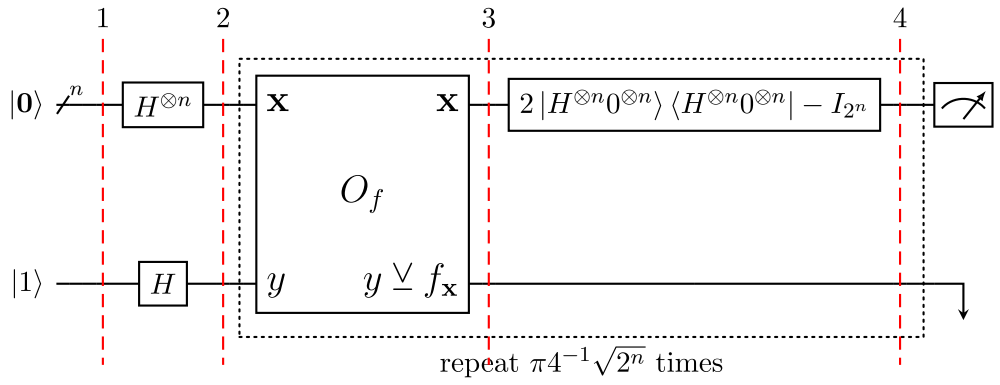
$\mathbf{x}$	000	100	010	110	001	101	011	111
$f_{\mathbf{x}}$	0	0	0	0	0	1	0	0

The circuit of Grover's algorithm is presented in Figure 4. Once again, the algorithm can be broken down in four steps (indicated by the red dashed lines in the circuit). The first steps of Grover's algorithm are almost identical to those of the Deutsch-Josza algorithm. The only difference is that oracle O_f only kicks back the phase of the basis state that corresponds to the target entry. In our example, $|\mathbf{\Psi}_1\rangle = |0001\rangle$, $|\mathbf{\Psi}_2\rangle = |+++-\rangle$, and $|\mathbf{\Psi}_3\rangle$ is given by (the basis state whose phase has been kicked back is indicated in blue):

$$|\mathbf{\Psi}_3\rangle = \frac{1}{\sqrt{8}}\left(|000\rangle + |100\rangle + |010\rangle + |110\rangle + |001\rangle - |101\rangle + |011\rangle + |111\rangle\right) \otimes |-\rangle\,.$$

At this stage, if we observe the first three qubits, there is an equal probability that the system collapses into either of the 8 basis states. In other words, we have 1 chance out of 8 to find out that $|101\rangle$ is the target basis state; measuring the state of the system at this stage is equivalent to randomly guessing the target entry. To increase the probability of measuring the correct basis state,

Figure 4. Circuit of Grover's algorithm


1
2
3
4
$|\mathbf{0}\rangle$
n
$H^{\otimes n}$
$\mathbf{x}$
$\mathbf{x}$
O_f
$2\,|H^{\otimes n}0^{\otimes n}\rangle\langle H^{\otimes n}0^{\otimes n}| - I_{2^n}$
$|1\rangle$
H
y
$y \veebar f_{\mathbf{x}}$
repeat $\pi 4^{-1}\sqrt{2^n}$ times

Grover uses a so-called "difusion" operator. In short, the diffusion operator is a quantum gate that reflects the probability amplitudes of the basis states about the average probability amplitude. The goal of the diffusion operator is to amplify the probability amplitude of the target basis state. For three qubits, the diffusion operator is given by:

$$2\left|H^{\otimes 3}0^{\otimes 3}\right\rangle\left\langle H^{\otimes 3}0^{\otimes 3}\right| - I_8 = \begin{pmatrix} -0.75 & 0.25 & 0.25 & 0.25 & 0.25 & 0.25 & 0.25 & 0.25 \\ 0.25 & -0.75 & 0.25 & 0.25 & 0.25 & 0.25 & 0.25 & 0.25 \\ 0.25 & 0.25 & -0.75 & 0.25 & 0.25 & 0.25 & 0.25 & 0.25 \\ 0.25 & 0.25 & 0.25 & -0.75 & 0.25 & 0.25 & 0.25 & 0.25 \\ 0.25 & 0.25 & 0.25 & 0.25 & -0.75 & 0.25 & 0.25 & 0.25 \\ 0.25 & 0.25 & 0.25 & 0.25 & 0.25 & -0.75 & 0.25 & 0.25 \\ 0.25 & 0.25 & 0.25 & 0.25 & 0.25 & 0.25 & -0.75 & 0.25 \\ 0.25 & 0.25 & 0.25 & 0.25 & 0.25 & 0.25 & 0.25 & -0.75 \end{pmatrix},$$

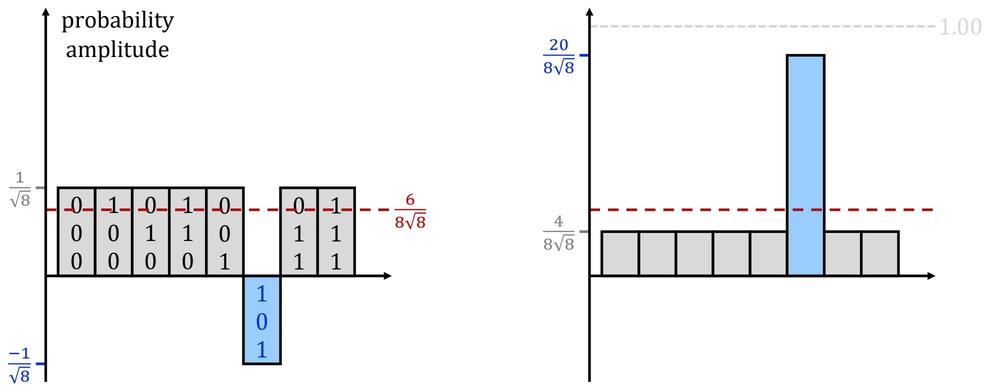
where $|0^{\otimes n}\rangle$ is the n^{th} tensor power of $|0\rangle$, $|v\rangle\langle f|$ is the outer product of $|v\rangle$ and $\langle f|$, and I_{2^n} is an identity matrix of dimension $2^n \times 2^n$. When applied to our example, one can verify that the average probability amplitude is $6(8\sqrt{8})^{-1}$, and we get:

$$|\mathbf{\Psi}_4\rangle = \left(\frac{20}{8\sqrt{8}}|101\rangle + \frac{4}{8\sqrt{8}}\left(|000\rangle + |100\rangle + |010\rangle + |110\rangle + |001\rangle + |011\rangle + |111\rangle\right)\right) \otimes |-\rangle\,.$$

In other words, the probability amplitude of the target basis state (i.e., $-(8\sqrt{8})^{-1}$) is reflected about $6(8\sqrt{8})^{-1}$, resulting in probability amplitude $20(8\sqrt{8})^{-1}$. The probability amplitudes of all other basis states (i.e., $(8\sqrt{8})^{-1}$) are also reflected, resulting in probability amplitudes $4(8\sqrt{8})^{-1}$. These reflections are illustrated in Figure 5.

After a first iteration of steps 3 and 4, the probability of identifying the correct target state has increased from 0.125 to 0.7813 (i.e., $(20(8\sqrt{8})^{-1})^2$). We can further increase this probability

Figure 5. Probability amplitudes of the different basis states before and after applying the diffusion operator in a first iteration (target basis state is printed in blue, other basis states are printed in grey, and the average amplitude used by the diffusion operator is printed in red).



by repeating steps 3 and 4. We get:

$$|\mathbf{\Psi}_{3.2}\rangle = \left(-\frac{20}{8\sqrt{8}}|101\rangle + \frac{4}{8\sqrt{8}}\left(|000\rangle + |100\rangle + |010\rangle + |110\rangle + |001\rangle + |011\rangle + |111\rangle\right)\right) \otimes |-\rangle\,,$$

resulting in an average probability amplitude of $(8\sqrt{8})^{-1}$, and:

$$|\mathbf{\Psi}_{4.2}\rangle = \left(\frac{22}{8\sqrt{8}}|101\rangle - \frac{2}{8\sqrt{8}}\left(|000\rangle + |100\rangle + |010\rangle + |110\rangle + |001\rangle + |011\rangle + |111\rangle\right)\right) \otimes |-\rangle\,.$$

After a second iteration of steps 3 and 4, the probability of success increases to 0.9453; we are 94.53 percent certain to identify the target basis state after only two calls to function f. Another repetition of steps 3 and 4, however, will not increase the success probability:

$$|\mathbf{\Psi}_{3.3}\rangle = \left(-\frac{22}{8\sqrt{8}}|101\rangle - \frac{2}{8\sqrt{8}}\left(|000\rangle + |100\rangle + |010\rangle + |110\rangle + |001\rangle + |011\rangle + |111\rangle\right)\right) \otimes |-\rangle\,,$$

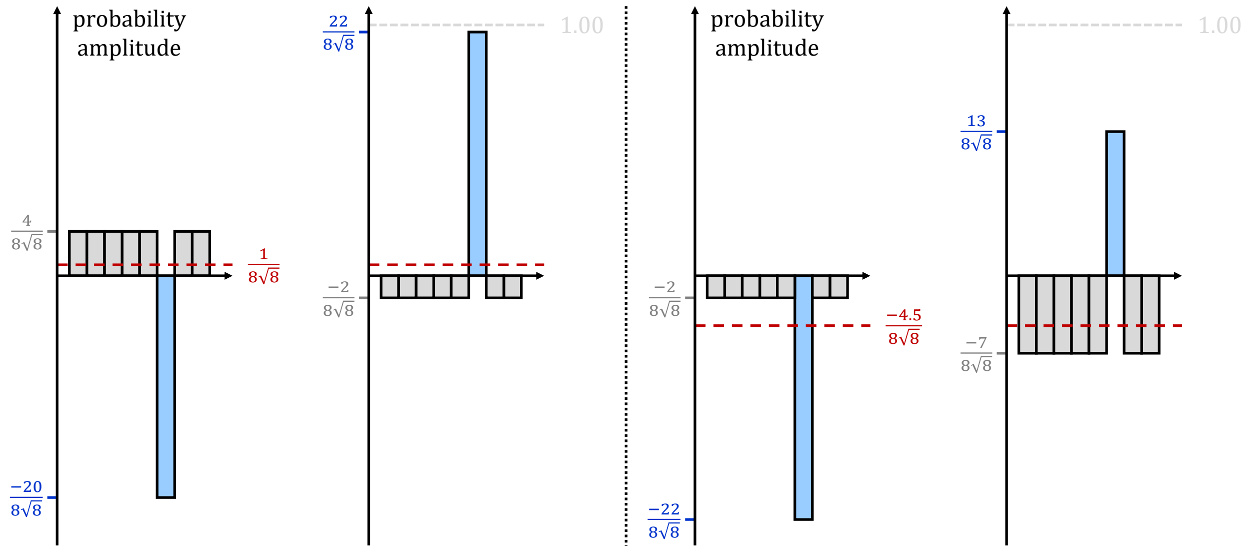
resulting in an average probability amplitude of $-4.5(8\sqrt{8})^{-1}$, and:

$$|\mathbf{\Psi}_{4.3}\rangle = \left(\frac{13}{8\sqrt{8}}|101\rangle - \frac{7}{8\sqrt{8}}\left(|000\rangle + |100\rangle + |010\rangle + |110\rangle + |001\rangle + |011\rangle + |111\rangle\right)\right) \otimes |-\rangle\,.$$

After a third iteration of steps 3 and 4, the probability of success drops to 0.3301. Figure 6 illustrates the second and third iteration.

For different numbers of iterations, Table 2 lists the average probability amplitude, the probability amplitudes of the target and other states, and the probability to successfully identify the target basis state before the diffusion operator is applied. In addition, Figure 7 shows the probability amplitude of the target basis state after each iteration. From Figure 7, one can observe that the probability amplitude of the target basis state approximates a sine function that has its first extremum after two iterations. In fact, to find a target entry in a database of 2^n entries, Grover

Figure 6. Probability amplitudes of the different basis states before and after applying the diffusion operator in a second (left panel) and third (right panel) iteration (target basis state is printed in blue, other basis states are printed in grey, and the average amplitude used by the diffusion operator is printed in red).



has shown that the probability amplitude of the target basis state reaches a first extremum after approximately $\pi 4^{-1}\sqrt{2^n}$ iterations. In other words, whereas a classical algorithm requires 2^{n-1} calls to function f (on average), Grover's algorithm only needs $\pi 4^{-1}\sqrt{2^n}$ calls, resulting in a quadratic speedup for large n. In addition, for this problem, Bennett et al. (1997) have shown that it is not possible to achieve a better speedup on a quantum computer. Admittedly, the output of Grover's algorithm is probabilistic (as is the case with most quantum algorithms). However, for sufficiently large n, even if we have to repeat Grover's algorithm a number of times, it is still faster than a classical algorithm.

Note that, even if there is more than one target entry, Grover's algorithm can still be used to quickly identify one of the target entries. In general, if there are m target entries in a database of 2^n entries, Grover's algorithm needs approximately $\pi 4^{-1}\sqrt{m^{-1}2^n}$ iterations (again, this result cannot be improved on a quantum computer; an algorithm to calculate the exact number of required iterations is available in Appendix). If the number of target entries is unknown, Grover's algorithm can be evaluated for different values of m until a matching entry has been found. One approach is to first evaluate Grover's algorithm assuming $m = 2^n$, and to evaluate $m = m/2$ in subsequent runs until a matching entry is found.

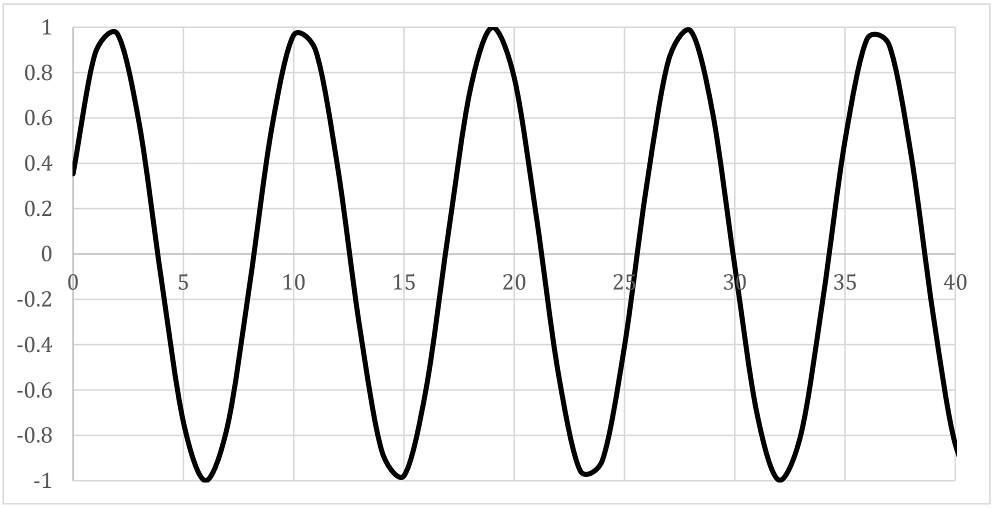
6. Application of Grover: Knapsack

In the previous section, Grover used a function f that outputs 1 if a set of n binary inputs corresponds to a given target entry in an unstructured database of 2^n entries. We can, however, also equip Grover's algorithm with other functions. By doing so, we can use Grover's algorithm to solve many classical problems. In this section, we use Grover's algorithm to solve the binary

Table 2. Average probability amplitude, probability amplitude of the target basis state, probability amplitude of the other basis states, and probability to successfully identify the target basis state before step 4 for different numbers of iterations.

iteration	average amplitude	target amplitude	other amplitude	success probability
1	0.2652	-0.3536	0.3536	0.1250
2	0.0442	-0.8839	0.1768	0.7813
3	-0.1989	-0.9723	-0.0884	0.9453
4	-0.3425	-0.5745	-0.3094	0.3301
5	-0.3149	0.1105	-0.3757	0.0122
6	-0.1298	0.7403	-0.2541	0.5480
7	0.1202	0.9999	-0.0055	0.9998
8	0.3100	0.7596	0.2458	0.5770
9	0.3449	0.1395	0.3743	0.0195
10	0.2073	-0.5504	0.3156	0.3029
11	-0.0339	-0.9650	0.0991	0.9313
12	-0.2582	-0.8972	-0.1669	0.8049
13	-0.3534	-0.3807	-0.3495	0.1450
14	-0.2719	0.3261	-0.3573	0.1063
15	-0.0544	0.8698	-0.1865	0.7566
16	0.1902	0.9787	0.0776	0.9578

Figure 7. Probability amplitude of the target basis state after each iteration.



knapsack problem (we have chosen this problem because it is a typical OR problem that can be easily implemented as a function to be used in Grover's algorithm; other OR problems, however, can be implemented just as well).

The binary knapsack problem is an NP-complete problem that has been studied for more than a century (see e.g., Kellerer et al. (2004)). The goal of the binary knapsack problem is to maximize the value of a knapsack by adding items from a pool of n items. Each item i has a weight w_i and a value v_i. The weight of the selected items should not exceed W, the maximum weight capacity of the knapsack. The binary knapsack problem may be formulated as follows:

$$\text{Maximize} \quad \sum_{i=1}^{n} v_i x_i \tag{3}$$

$$\text{s.t.} \quad \sum_{i=1}^{n} w_i x_i \leq W \tag{4}$$

$$x_i \in \{0, 1\}, \quad \forall\, 1 \leq i \leq n \tag{5}$$

In this formulation, x_i is a binary decision variable that determines whether or not item i has been added to the knapsack, objective function 3 maximizes the value of the knapsack, constraint 4 ensures that the items in the knapsack do not exceed the maximum weight, and constraint 5 ensures that decision variables are binary.

To solve the binary knapsack problem, we equip Grover's algorithm with a function $f_{\mathbf{x}}$ that outputs 1 if knapsack $\mathbf{x} = \{x_1, \ldots, x_n\}$ is valid (and 0 otherwise). As a result, Grover's algorithm will only kickback the phase of basis state $|\mathbf{x}\rangle$ if knapsack $\mathbf{x}$ is valid. Knapsack $\mathbf{x}$ is valid if it is feasible (i.e., if it satisfies constraint 4), and if it has a value of at least V. To ensure that knapsack $\mathbf{x}$ has a value of at least V, we impose an additional constraint:

$$\sum_{i=1}^{n} v_i x_i \geq V \tag{6}$$

We use a binary search procedure to find the optimal knapsack and its value (denoted x^* and V^*, respectively). An outline of the procedure is provided in Algorithm 1 (note that, with a minimum of adaptations, this procedure can also be used to solve other OR problems). After initializing a minimum knapsack value $V_{\min}$ and a maximum knapsack value $V_{\max}$, the procedure evaluates at most $\lceil \log_2(1 + (V_{\max} - V_{\min})(\min_i v_i)^{-1}) \rceil$ iterations of an outer loop. In each iteration of the outer loop, we initialize a knapsack value V and the number of valid solutions m. V is initialized as the largest value that is divisible by $\min_i v_i$, and that is still smaller-than-or-equal-to the center of the interval that has endpoints $V_{\min}$ and $V_{\max}$. In addition, we let $m = 2^n$ because we don't know the number of valid solutions (as also explained at the end of the previous section). Next, we perform at most n iterations of an inner loop. In each of these iterations, we perform $\pi 4^{-1} \sqrt{m^{-1} 2^n}$ repetitions of Grover's algorithm, and measure the basis state into which the system collapses (i.e., we measure $|\mathbf{x}\rangle$). Because the outcome of Grover's algorithm is probabilistic, it is possible that solution $\mathbf{x}$ is not valid. In addition, if no valid solution exists, any basis state that we measure will correspond to an invalid solution. Therefore, we need to verify whether $\mathbf{x}$ is valid; we need to verify whether $\sum_{i=1}^{n} w_i x_i \leq W$ and $\sum_{i=1}^{n} v_i x_i \geq V$. If the validity of solution $\mathbf{x}$ has been verified, $V_{\min}$ is updated (i.e., $V_{\min} = V + \min_i v_i$), we break the inner loop, and start a new iteration of the outer loop in which we reinitialize V and m. If, on the other hand, solution $\mathbf{x}$ is not valid, there are two options. If $m > 1$, we let $m = m/2$, and perform another iteration of the inner loop. If, however, we already evaluated all values of m, we update $V_{\max}$ (i.e., $V_{\max} = V - \min_i v_i$), we break the inner loop, and start a new iteration of the outer loop in which we reinitialize V and m. This process continues until $V_{\min} > V_{\max}$. The last found solution and its value are optimal. Note that, in order to reduce the number of iterations, lower and upper bound procedures may be used to initialize $V_{\min}$ and $V_{\max}$.

Algorithm 1: Procedure to solve the binary knapsack problem using Grover's algorithm

```
initialize $V_{\min} = 0$ and $V_{\max} = \sum_{i=1}^{n} v_i$;
do
    $V = \min_i v_i \left\lfloor (2\min_i v_i)^{-1}(V_{\min} + V_{\max}) \right\rfloor$;
    $m = 2^n$;
    do
        perform $\pi 4^{-1}\sqrt{m^{-1}2^n}$ repetitions of Grover's algorithm equipped with $f_{\mathbf{x}}$;
        measure basis state $|\mathbf{x}\rangle$;
        if validity of solution $\mathbf{x}$ has been verified then
            update optimal solution $\mathbf{x}^* = \mathbf{x}$ and its value $V^* = V$;
            $V_{\min} = V + \min_i v_i$;
            break;
        else if $m > 1$ then
            $m = m/2$;
        else
            $V_{\max} = V - \min_i v_i$;
            break;
    while $m \geq 1$;
while $V_{\min} \leq V_{\max}$;
```

The circuit of function $f_{\mathbf{x}}$ is presented in Figure 8. The circuit may be divided into three parts (indicated by the red dashed lines in the circuit). In the first two parts, we determine whether knapsack $\mathbf{x}$ satisfies constraints 4 and 6, respectively. In a third part, we obtain the output of function $f_{\mathbf{x}}$. In what follows, we use an example to demonstrate the dynamics of the circuit. In the example, we have a pool of three items and a knapsack that has maximum weight capacity $W = 4$. The weights and values of the items are listed in Table 3. Since we are working with qubits, all weights and values need to be converted to binary values. Let b denote the number of qubits that are required to represent the largest weight or value used in the example. In our example, $b = 3$, and the qubits that represent the maximum weight capacity are initialized as follows: $\left|c_1^W\right\rangle = |1\rangle$, $\left|c_2^W\right\rangle = |0\rangle$, and $\left|c_3^W\right\rangle = |0\rangle$ (with $\mathbf{c}^W = \left\{c_1^W, \ldots, c_b^W\right\}$). The binary conversions of the

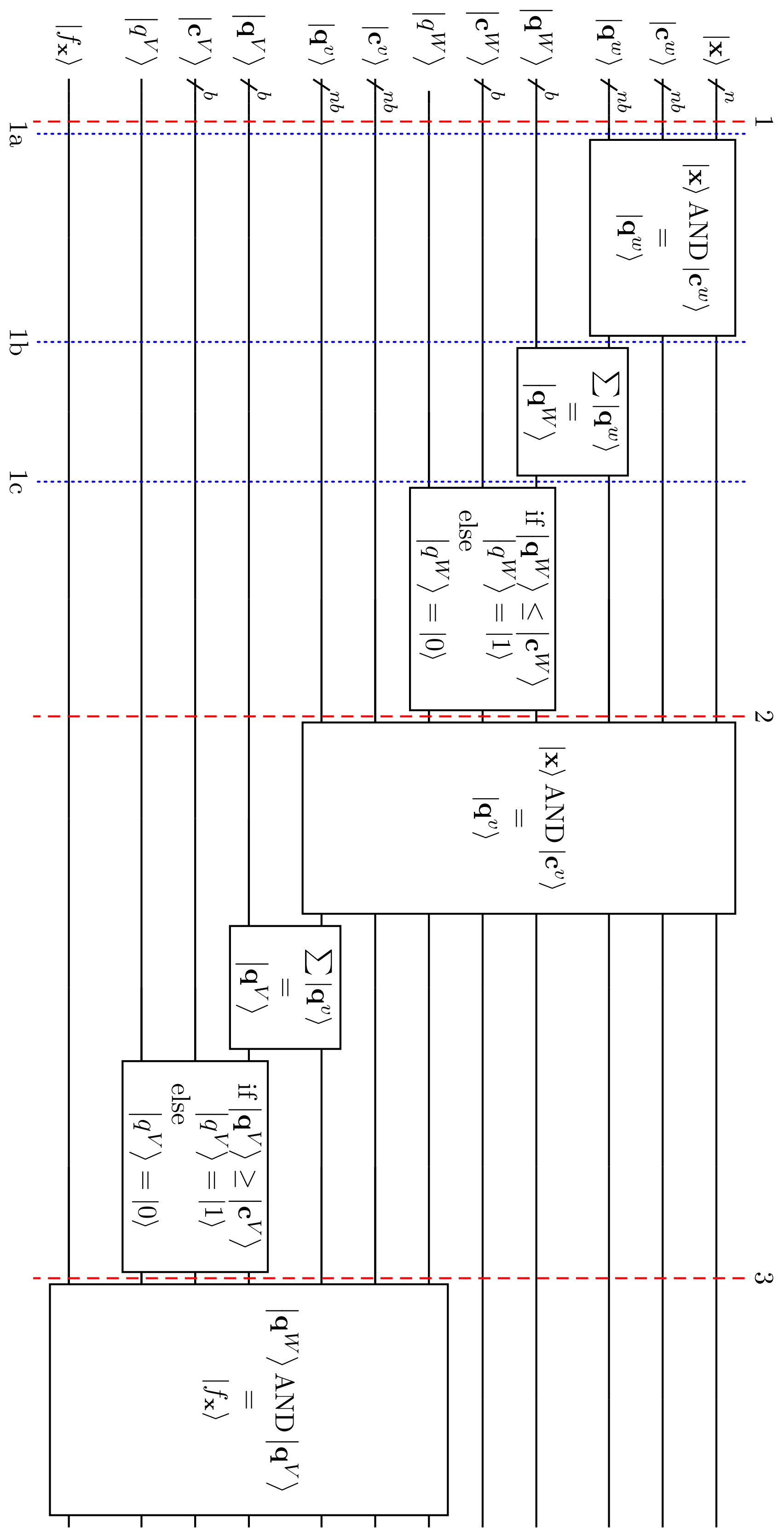

$|\mathbf{x}\rangle$ n
$|\mathbf{c}^w\rangle$ nb
$|\mathbf{q}^w\rangle$ nb
$|\mathbf{q}^W\rangle$ b
$|\mathbf{c}^W\rangle$ b
$|q^W\rangle$
$|\mathbf{c}^v\rangle$ nb
$|\mathbf{q}^v\rangle$ nb
$|\mathbf{q}^V\rangle$ b
$|\mathbf{c}^V\rangle$ b
$|q^V\rangle$
$|f_\mathbf{x}\rangle$
1
1a
1b
1c
2
3
$|\mathbf{x}\rangle$ AND $|\mathbf{c}^w\rangle$ = $|\mathbf{q}^w\rangle$
$\sum|\mathbf{q}^w\rangle$ = $|\mathbf{q}^W\rangle$
if $|\mathbf{q}^W\rangle \leq |\mathbf{c}^W\rangle$ $|q^W\rangle = |1\rangle$ else $|q^W\rangle = |0\rangle$
$|\mathbf{x}\rangle$ AND $|\mathbf{c}^v\rangle$ = $|\mathbf{q}^v\rangle$
$\sum|\mathbf{q}^v\rangle$ = $|\mathbf{q}^V\rangle$
if $|\mathbf{q}^V\rangle \geq |\mathbf{c}^V\rangle$ $|q^V\rangle = |1\rangle$ else $|q^V\rangle = |0\rangle$
$|\mathbf{q}^W\rangle$ AND $|\mathbf{q}^V\rangle$ = $|f_\mathbf{x}\rangle$

Figure 8. Circuit of knapsack function $f_\mathbf{x}$

weights and values of the items are listed in Table 3, where $\mathbf{c}^w = \{\mathbf{c}_1^w, \ldots, \mathbf{c}_n^w\}$, $\mathbf{c}^v = \{\mathbf{c}_1^v, \ldots, \mathbf{c}_n^v\}$, $\mathbf{c}_i^w = \{c_{i1}^w, \ldots, c_{ib}^w\}$, and $\mathbf{c}_i^v = \{c_{i1}^v, \ldots, c_{ib}^v\}$. In our example, $\left|\mathbf{c}^W\right\rangle$, $|\mathbf{c}^w\rangle$, and $|\mathbf{c}^v\rangle$ are initialized as

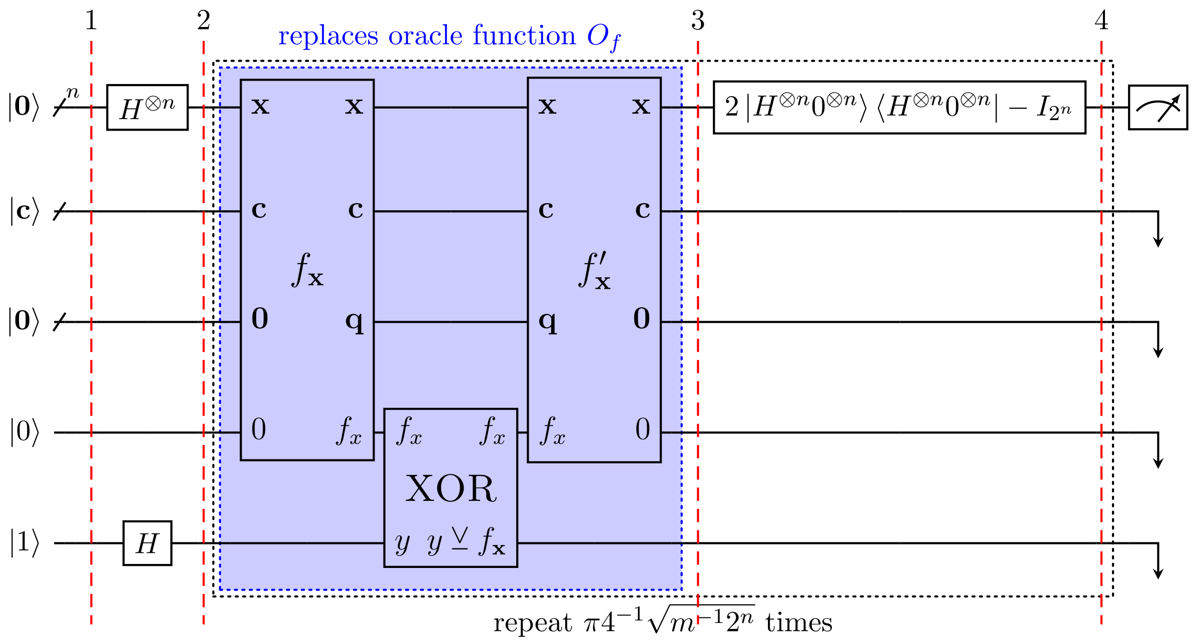
Table 3. Data for example knapsack problem with three items

i	w_i	c_{i1}^w	c_{i2}^w	c_{i3}^w	v_i	c_{i1}^v	c_{i2}^v	c_{i3}^v
1	2	0	1	0	3	0	1	1
2	3	0	1	1	1	0	0	1
3	2	0	1	0	2	0	1	0

$|100\rangle$, $|010011010\rangle$, and $|011001010\rangle$, respectively. Next, imagine that we use the circuit of function $f_{\mathbf{x}}$ to evaluate knapsack $\mathbf{x} = \{1, 0, 1\}$. In this case, $|\mathbf{x}\rangle$ is initialized as $|101\rangle$. In a first part of the circuit, we distinguish three steps (indicated by the blue dotted lines in the circuit). In a first step (1a), we determine how much weight each item contributes to knapsack $\mathbf{x}$. The contributed weight of the items is represented by $\mathbf{q}^w = \{\mathbf{q}_1^w, \ldots, \mathbf{q}_n^w\}$, with $\mathbf{q}_i^w = \{q_{i1}^w, \ldots, q_{ib}^w\}$. $|\mathbf{q}^w\rangle$ is initialized as $|\mathbf{0}\rangle$, and is used to store the result of an AND operation on $|\mathbf{x}\rangle$ and $|\mathbf{c}^w\rangle$. In our example, one can verify that $|\mathbf{q}^w\rangle = |010000010\rangle$. In a second step (1b), we obtain the weight of knapsack $\mathbf{x}$ itself (represented by $\mathbf{q}^W = \left\{q_1^W, \ldots, q_b^W\right\}$). $\left|\mathbf{q}^W\right\rangle$ is initialized as $|\mathbf{0}\rangle$, and is used to store the result of $\sum_{i=1}^n |\mathbf{q}_i^w\rangle$. To perform the aforementioned addition, we use a circuit that was developed by Draper (2002). In our example, $\left|\mathbf{q}^W\right\rangle = |100\rangle$. As a last step (1c), we verify whether knapsack $\mathbf{x}$ satisfies constraint 4. This can be done using the quantum equivalent of a digital magnitude comparator (see e.g., Xia et al (2018)). The digital magnitude comparator compares the weight of knapsack $\mathbf{x}$ (represented by $\mathbf{q}^W$) and the maximum knapsack weight (represented by $\mathbf{c}^W$), and outputs 1 if a smaller-than-or-equal-to condition is met. The result is stored in a qubit that is initialized as $\left|q^W\right\rangle = |0\rangle$. In our example, constraint 4 is satisfied, and $\left|q^W\right\rangle$ evaluates to $|1\rangle$. This concludes the first part of the circuit. The second part of the circuit is almost identical to the first part, and verifies whether the value of the knapsack (stored in $\left|\mathbf{q}^V\right\rangle = \sum_{i=1}^n |\mathbf{q}_i^v\rangle$, where $\mathbf{q}^V = \left\{q_1^V, \ldots, q_b^V\right\}$, $\mathbf{q}^v = \{\mathbf{q}_1^v, \ldots, \mathbf{q}_n^v\}$, and $\mathbf{q}_i^v = \{q_{i1}^v, \ldots, q_{ib}^v\}$) is at least equal to value V (stored in $\left|\mathbf{c}^V\right\rangle$, where $\mathbf{c}^V = \left\{c_1^V, \ldots, c_b^V\right\}$). If this is the case, constraint 6 is satisfied, and $\left|q^V\right\rangle = |1\rangle$. If, however, knapsack $\mathbf{x}$ does not satisfy constraint 6, $\left|q^V\right\rangle = |0\rangle$. In a last part of the circuit, we obtain the outcome of function $f_{\mathbf{x}}$ itself (stored in a qubit that is initialized as $|f_{\mathbf{x}}\rangle = |0\rangle$). If both constraints are satisfied, $|f_{\mathbf{x}}\rangle = |1\rangle$ (and $|f_{\mathbf{x}}\rangle = |0\rangle$ otherwise).

Figure 9 presents the circuit of Grover's algorithm equipped with function $f_{\mathbf{x}}$ (once more, the steps of the algorithm are indicated by red dashed lines). In this circuit, $\mathbf{x}$, $\mathbf{c}$, $\mathbf{q}$, and $f_{\mathbf{x}}$ are the qubits that are used to store the decision variables, the constant values that are used by function $f_{\mathbf{x}}$, the outcomes of operations performed by function $f_{\mathbf{x}}$, and the result of function $f_{\mathbf{x}}$, respectively. In addition, y is a qubit that is used to induce phase kickback. $|y\rangle$ is initialized as $|1\rangle$. The initialization of $|\mathbf{c}\rangle$ depends on the constant values that are used by function $f_{\mathbf{x}}$. All other qubits are initialized as $|0\rangle$. The gates grouped in blue replace oracle function O_f in the original circuit of Grover's algorithm (see Figure 4). These gates ensure that: (1) the phase of each target basis state is kicked back, and (2) $|\mathbf{q}\rangle$ and $|f_{\mathbf{x}}\rangle$ are reinitialized. To reinitialize $|\mathbf{q}\rangle$ and $|f_{\mathbf{x}}\rangle$, it suffices to apply all gates of function $f_{\mathbf{x}}$ in reverse order and to use Draper's circuit to subtract rather than to add (note that applying all gates of function $f_{\mathbf{x}}$ in reverse order does not impact $|\mathbf{x}\rangle$ nor $|\mathbf{c}\rangle$; in addition, after step 2, qubit y remains in state $|-\rangle$ until completion of the algorithm). $|\mathbf{q}\rangle$ and $|f_{\mathbf{x}}\rangle$ need to be reinitialized because steps 3 and 4 are repeated $\pi 4^{-1}\sqrt{m^{-1}2^n}$ times (and for each repetition of Grover's algorithm, $|\mathbf{q}\rangle$ and $|f_{\mathbf{x}}\rangle$ need to be initialized as $|\mathbf{0}\rangle$ and $|0\rangle$). Alternatively, if we have sufficient qubits at our disposal, we don't need to "recycle" qubits, and we can just use new qubits for each repetition of Grover's algorithm. Contemporary quantum computers, however, only have a limited number of qubits, and hence, recycling qubits is necessary.

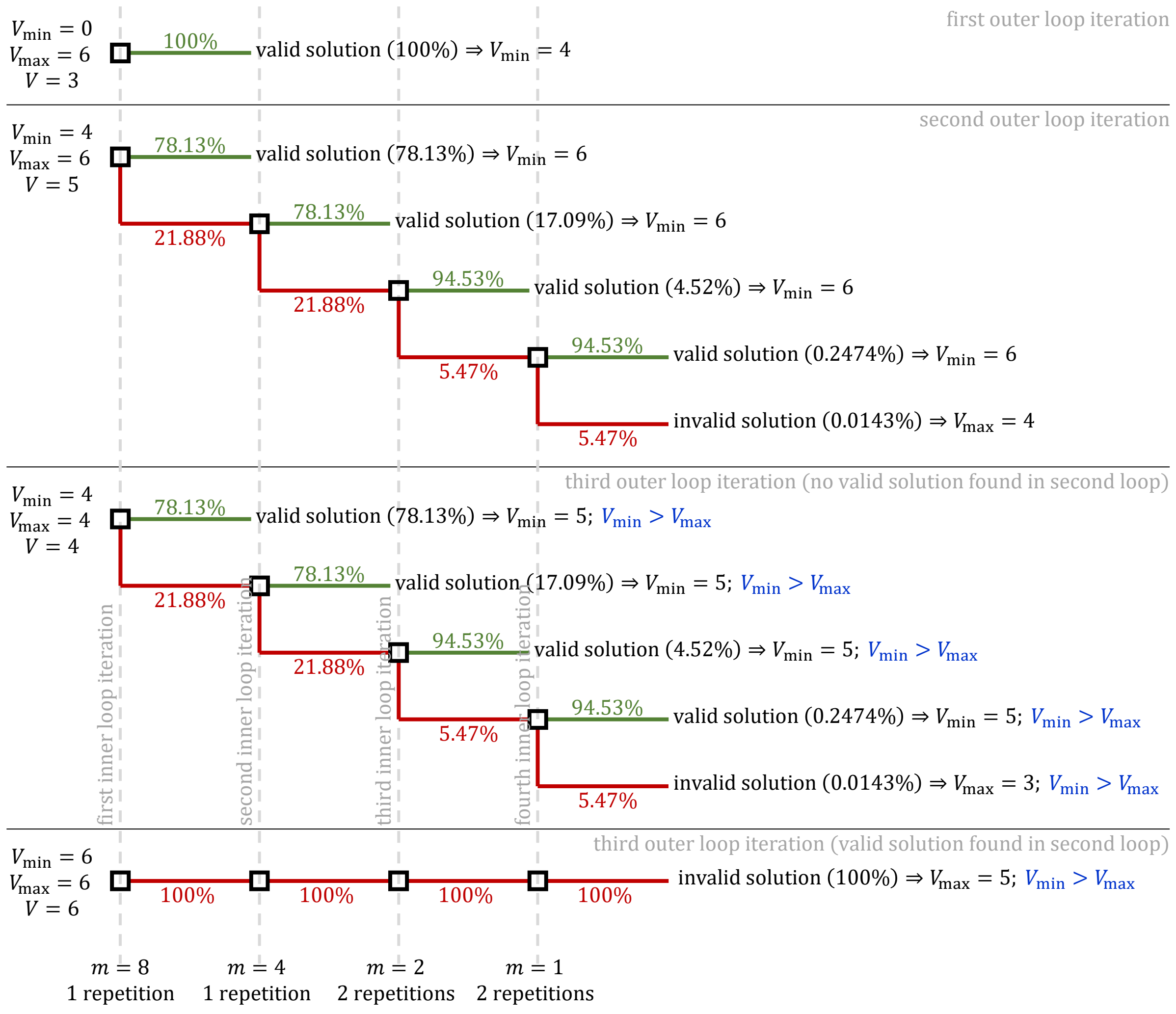
Figure 9. Circuit of Grover's algorithm equipped with function $f_{\mathbf{x}}$



We can now use Algorithm 1 to find the optimal solution to our example knapsack problem. The steps of the algorithm are illustrated in Figure 10. First, we let $V_{\min} = 0$, and $V_{\max} = \sum_{i=1}^{n} v_i = 6$. Next, we initiate a first iteration of the outer loop, and initialize $V = \min_i v_i \left\lfloor (2\min_i v_i)^{-1}(V_{\min} + V_{\max}) \right\rfloor = 3$ and $m = 2^n = 8$. In a first iteration of the inner loop, we evaluate $\pi 4^{-1}\sqrt{m^{-1}2^n} = 0.7854 \approx 1$ repetition of Grover's algorithm. Given $V = 3$, there are two valid solutions (i.e., solution $\{1,0,0\}$ and $\{1,0,1\}$). As such, the phases of basis states $|100\rangle$ and $|101\rangle$ are kicked back by Grover's algorithm. One can verify that, after a single repetition of Grover's algorithm, we have a 50 percent probability to measure $|100\rangle$, and a 50 percent probability to measure $|101\rangle$. In other words, if we observe the system after one repetition of Grover's algorithm, we are certain to measure a valid solution that has at least value $V = 3$. After verifying that $\{1,0,0\}$ (or $\{1,0,1\}$) is a valid solution, we let $V_{\min} = V + \min_i v_i = 4$, update the optimal solution and its value, break the inner loop, and start a second iteration of the outer loop. In this second iteration, we let $V = 5$ and $m = 8$. Next, we initiate a first iteration of the inner loop, and evaluate $\pi 4^{-1}\sqrt{m^{-1}2^n} = 0.7854 \approx 1$ repetition of Grover's algorithm. Given $V = 5$, however, one can verify that there is only one valid solution (i.e., solution $\{1, 0, 1\}$). If there is only a single valid solution, a single repetition of Grover's algorithm only has a 78.13 percent chance to measure the correct basis state (see also the previous section). In other words, there is a 21.88 percent chance that we measure a basis state that is not valid. This illustrates why every solution resulting from Grover's algorithm needs to be verified. If the validity of the solution has been verified, we update the optimal solution, let $V_{\min} = 6$, break the inner loop, and start a third iteration of the outer loop in which $V = 6$ (in this third iteration, no valid solution can be found for any value of m, and we let $V_{\max} = V - \min_i v_i = 5$; as a result, $V_{\min} > V_{\max}$, and the procedure stops). If, on the other hand, no valid solution was identified (with probability 0.2188), we let $m = m/2 = 4$, and we perform $\pi 4^{-1}\sqrt{m^{-1}2^n} = 1.1107 \approx 1$ repetition of Grover's algorithm. Again, we have a 78.13 percent chance to measure the basis state that corresponds to valid solution $\{1, 0, 1\}$. As such, we have a 17.09 percent chance to end up a situation where we did not measure a valid solution in the first iteration of the inner loop, but did measure a valid solution in the second iteration of the inner loop. If a valid solution was measured, we update the optimal solution, let $V_{\min} = 6$, break the inner loop, and start a third iteration of the outer loop in which $V = 6$. If no valid solution was measured, we let $m = 2$, and perform $\pi 4^{-1}\sqrt{m^{-1}2^n} = 1.5708 \approx 2$ repetitions of Grover's algorithm. With 2 repetitions of Grover's algorithm, we have a 94.53 percent chance to identify a valid solution. If, however, we still fail to identify a valid solution, we let $m = 1$, and again perform 2 repetitions of Grover's algorithm. In total, we are 99.99 percent certain to measure $|101\rangle$, and to obtain $\{1, 0, 1\}$ as a valid solution during the second iteration of the outer loop. There is, however, a 0.0143 percent probability to measure the wrong basis state. If this happens, verification of the solution fails, we cannot reduce m any further, and we let $V_{\max} = V - \min_i v_i = 4$ (i.e., we have wrongfully concluded that $V = 5$ has no valid solution; this illustrates why it is possible that we are unable to identify the optimal solution if we use Grover's algorithm in an optimization procedure). At this point, we enter a third iteration with $V = 4$ (here again, we have a 99.99 percent probability to identify $\{1, 0, 1\}$ as the only valid solution, after which we update $V_{\min}$ such that $V_{\min} > V_{\max}$, and the procedure stops; conversely, we have a 0.0143 percent probability to identify an invalid solution, after which we update $V_{\max}$ such that $V_{\min} > V_{\max}$, and the procedure also stops). Even though we are reasonably certain to find the optimal solution, we also require at least 8 repetitions of Grover's algorithm (and at most 13). In addition, we need to verify the validity of at least 6 knapsack solutions (and at most 9; note that a brute-force classical approach would need to evaluate at most 8 knapsack solutions).

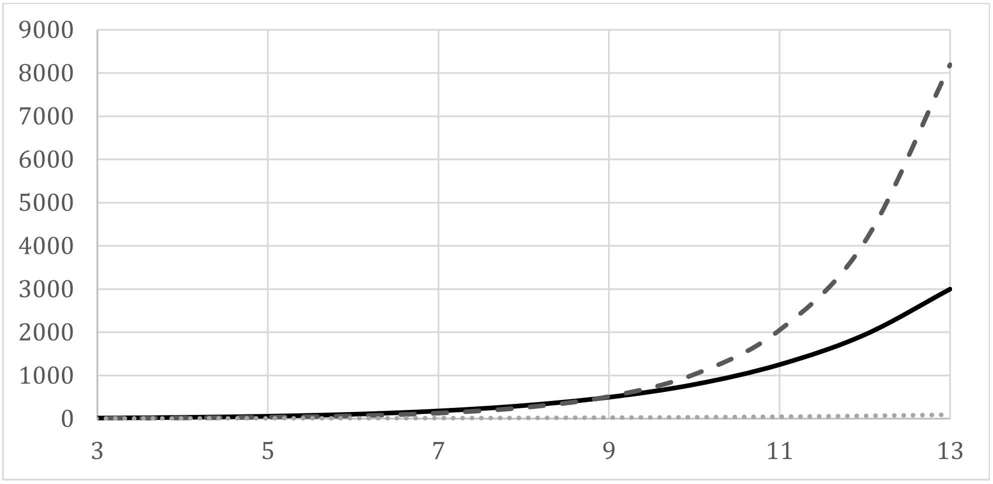
In order to verify the required number of repetitions of Grover's algorithm for various values of n, we perform an additional experiment. In this experiment, we use Algorithm 1 to solve a knapsack problem that has a pool of n items. For the purpose of this experiment, we let $V_{\min} = 0$ and $V_{\max} = 2^n$ at the start of the algorithm, and assume that there is only a single valid solution that has value 0.5×2^n. As a result, if we evaluate a value $V <= 0.5 \times 2^n$, we can find 1 valid solution. If, on the other hand, we evaluate a value $V > 0.5 \times 2^n$, no valid solution can be found. Figure 11 summarizes the result of the experiment, and shows the expected number of repetitions of Grover's

Figure 10. Illustration of the dynamics of Algorithm 1 for the example knapsack problem. In each node (represented by a square), we perform 1 to 2 repetitions of Grover's algorithm (depending on the value of m), we measure the basis state, and we verify whether the corresponding solution is valid. If the solution is valid (indicated in green), $V_{\min}$ is updated. If, on the other hand, the solution is not valid (indicated in red), we update m or $V_{\max}$ (if $m = 1$). The procedure stops if $V_{\min} > V_{\max}$ (indicated in blue).



first outer loop iteration
$V_{\min} = 0$ $V_{\max} = 6$ $V = 3$
100%
valid solution (100%) ⇒ $V_{\min} = 4$
second outer loop iteration
$V_{\min} = 4$ $V_{\max} = 6$ $V = 5$
78.13%
valid solution (78.13%) ⇒ $V_{\min} = 6$
21.88%
78.13%
valid solution (17.09%) ⇒ $V_{\min} = 6$
21.88%
94.53%
valid solution (4.52%) ⇒ $V_{\min} = 6$
5.47%
94.53%
valid solution (0.2474%) ⇒ $V_{\min} = 6$
5.47%
invalid solution (0.0143%) ⇒ $V_{\max} = 4$
third outer loop iteration (no valid solution found in second loop)
$V_{\min} = 4$ $V_{\max} = 4$ $V = 4$
78.13%
valid solution (78.13%) ⇒ $V_{\min} = 5$; $V_{\min} > V_{\max}$
21.88%
78.13%
valid solution (17.09%) ⇒ $V_{\min} = 5$; $V_{\min} > V_{\max}$
21.88%
94.53%
valid solution (4.52%) ⇒ $V_{\min} = 5$; $V_{\min} > V_{\max}$
5.47%
94.53%
valid solution (0.2474%) ⇒ $V_{\min} = 5$; $V_{\min} > V_{\max}$
5.47%
invalid solution (0.0143%) ⇒ $V_{\max} = 3$; $V_{\min} > V_{\max}$
first inner loop iteration
second inner loop iteration
third inner loop iteration
fourth inner loop iteration
third outer loop iteration (valid solution found in second loop)
$V_{\min} = 6$ $V_{\max} = 6$ $V = 6$
100%
100%
100%
100%
invalid solution (100%) ⇒ $V_{\max} = 5$; $V_{\min} > V_{\max}$
$m = 8$ 1 repetition
$m = 4$ 1 repetition
$m = 2$ 2 repetitions
$m = 1$ 2 repetitions

Figure 11. Expected number of repetitions of Grover's algorithm (vertical axis) when solving a knapsack problem with n items using Algorithm 1 (black full line). For reference, function $f(n) = 2^n$ (dark grey dashed line) and $f(n) = \sqrt{2^n}$ (light grey dotted line) have also been added.



algorithm required to solve a knapsack problem with n items using Algorithm 1. One can observe that, as n increases, the number of required repetitions of Grover's algorithm increases rapidly. Similar results were obtained in other experiments (we discuss two additional experiments in the Appendix). We conclude that, for a knapsack with large n, computational gains can certainly be achieved when compared to a brute-force classical approach that evaluates 2^n knapsack solutions. However, we cannot claim that this speedup is quadratic.

7. Beyond Grover

In the previous section, we used Grover's algorithm to solve the knapsack problem, and have shown that the quadratic speedup (that is attributed to Grover's algorithm) was not achieved. A practical implementation of Grover's algorithm requires additional repetitions of the algorithm, and also needs to verify the validity of the measured solution after each run. In addition, the outcome of Grover's algorithm is probabilistic. As a result, there is no guarantee the algorithm will output the optimal solution. Even if we ignore these implementation challenges, Grover's algorithm has another major drawback. To illustrate this drawback, let's return to the initial purpose of Grover's algorithm: to find a target entry in an unstructured database of 2^n entries using a minimum number of calls to a lookup function f. Whereas a classical algorithm requires an average of 2^{n-1} calls to lookup function f, Grover's algorithm only needs $\sqrt{2^n}$ calls, resulting in a quadratic speedup for large n. However, if we consider a structured database, Grover's algorithm still needs $\sqrt{2^n}$ operations. A classical algorithm may need far less calls. For instance, if the database is sorted, binary search may be used, and $n+1$ calls suffice. In other words, if a classical algorithm can exploit the structure of a problem, it can easily outperform an implementation of Grover's algorithm.

In order to improve the performance of Grover's algorithm, Cerf et al. (2000) have suggested to use a "nested quantum search" to solve NP-complete problems. Rather than using Grover's algorithm to search the entire solution space to find the optimal solution (as we have done in the previous section), a nested quantum search "nests" one quantum search within another. This way, partial solutions (that are obtained efficiently using Grover's algorithm) are used to build other partial solutions that, in turn, can be used to build the optimal solution. The nested quantum search algorithm is the quantum counterpart of a classical nested search algorithm, and often allows to exploit the structure of most problems. A classical nested search algorithm performs $b^{\alpha n}$ operations, where n is the number of decision variables, b is the number of values that can be assigned to the decision variables, and α is some number less-or-equal-than 1 that depends on the problem. Cerf et al. (2000) have shown that the nested quantum search only needs $b^{0.5\alpha n}$ operations, once more resulting in a quadratic speedup. In addition, they show that α decreases with an increasing nesting depth; with an increased number of nesting levels. Whereas Cerf et al. (2000) alleviate one drawback of Grover's algorithm, they also introduce another: in each nested quantum search the number of valid solutions (m) is unknown. As a result, each combination of valid solutions for each level of nesting needs to be assessed in order to be sufficiently certain that the nested quantum search outputs the correct solution. For instance, consider a knapsack problem that has a pool of $n = 24$ items. Assume that we have three levels of nesting, and that in each of the nested quantum searches we perform Grover's algorithm on 8 items. As a result, for each nested search we have 4 possible values for m that need to be verified (m can be 8,4,2, and 1), resulting in $3^4 = 64$ combinations of possible values for m for 3 levels of nesting. As a first combination, we assume $m = 8$ at level 1, $m = 8$ at level 2, and $m = 8$ at level 3; if no valid solution is found, we evaluate a second combination, and assume $m = 8$ at level 1, $m = 8$ at level 2, and $m = 4$ at level 3. This process continues until a valid solution is found, or until we conclude that no valid solution exists. Note that, to find the optimal knapsack, we need to use the binary search procedure outlined in Algorithm 1. As a result, in this example, we need to evaluate up to 64 combinations of nested searches for each value of V.

Other attempts have also been made to improve the performance of Grover's algorithm by mixing classical and quantum algorithms. One notable example is the quantum dynamic programming algorithm proposed by Ambainis et al. (2018). In their paper, Ambainis et al. (2018) outline (among others) an approach to solve the travelling salesman problem (without implementing it). Their approach builds on the Bellman-Held-Karp algorithm (1962), and uses a recursion that adopts Grover's algorithm to find the shortest path for a subset of cities. If, however, Grover's algorithm is used to find the shortest path for each subset of cities, it is highly likely that it will output the wrong solution for at least some of these subsets (after all, the outcome of Grover's algorithm is probabilistic). As a result, the outcome of the quantum dynamic program is not likely to be optimal (especially for dynamic programs with large state spaces, for which you would like to obtain a quantum speedup in the first place). Last but not least, also in this approach, additional repetitions of Grover's algorithm are required because we don't know the number of valid solutions in advance.

It is clear that several challenges need to be overcome if we want to use Grover's algorithm to solve typical OR problems. This, however, should not come as a surprise since there is also no classical algorithm that excels in solving each and every OR problem. Perhaps then, we should wait for dedicated quantum algorithms that solve dedicated problems? The design of quantum algorithms, unfortunately, is not as straightforward (as is shown by the limited number of groundbreaking quantum algorithms that have been published). In addition, Bennett et al. (1997) have shown that a quadratic speedup (as obtained by Grover's algorithm) is the best we can hope for when performing an unstructured search on a quantum computer. This limitation is one of the main reasons why many researchers are sceptical about the potential of quantum computing to (significantly) outperform classical algorithms when solving difficult OR problems (a quadratic speedup does not transform exponential time to polynomial time; see e.g., Aaronson (2008)) for a discussion on the difficulty of solving NP-complete problems on a quantum computer).

The biggest obstacle, however, may not be the development of quantum algorithms, but the computers on which they run. Today we are in the era of "Noisy Intermediate Scale Quantum computers" (NISQ), a term proposed by Preskill in 2018. NISQ are quantum computing systems that are noisy, and that don't have an effective implementation of error correction codes. As a result, NISQ do not produce reliable output. NISQ systems also have a number of qubits in the order of dozens to thousands. Preskill estimates that a system would need at least 10^6 quantum bits in order to achieve fault tolerance computing. As of 2022 the current state-of-the-art NISQ, is the IBM 127-qubit "Eagle" processor. Recently (2022) IBM released an updated road map in which they expect to release a new "Osprey" 433-qubit quantum processor by the end of 2022 and a "Condor" 1121-qubit quantum processor by the end of 2023. Although this progress seems remarkable for such a small time frame, there is still a long way to go before we have a real universal quantum computer.

Despite all these limitations, quantum computing still has a lot of potential. Even though the design of quantum algorithms is not straightforward, efficient quantum algorithms for solving difficult OR problems may yet be developed. After all, Shor's algorithm shows that a dedicated quantum algorithm can be devised to solve a difficult OR problem in polynomial time (note, however, that we don't know whether the prime factorization problem is NP-complete, nor do we know whether an efficient classical algorithm exists that can solve the problem efficiently). In addition, given their probabilistic nature, quantum algorithms may be particularly useful in heuristic procedures, or for solving problems where it is less important that the optimal solution is obtained. Also in the field of quantum simulation and machine learning advances are being made, with potential use cases in the pharmaceutical, chemical, automotive, and financial industries (McKinsey, 2021). Next to gate-based quantum computing (discussed in this paper), there is also adiabatic quantum computing. Adiabatic quantum computing exploits the fact that in physics everything tends to seek a minimum energy state. Using adiabatic quantum computing, OR problems can be formulated as energy minimization problems. Starting from an initial state, quantum annealing is then used to evolve the quantum system towards a low-energy state. This low-energy state corresponds to the optimal or near-optimal solution of the problem (see e.g., Farhi (2014)).

8. Conclusion

In this paper, we investigate the potential of quantum computing from an OR perspective. For this purpose, we discuss a number of quantum algorithms, among which Grover's algorithm. Arguably, Grover's algorithm is the most important quantum algorithm as it achieves a quadratic speedup (when compared to classical algorithms), and can be used to solve many OR problems. We are the first to use Grover's algorithm to solve a classic OR problem, the knapsack problem, and verify our calculations using Qiskit (IBM's software development kit to simulate or operate a quantum computer). Our code for the knapsack problem and templates for solving other OR problems are made available as supplementary material. Note that, even though we focus on the knapsack problem, our insights are valid for other OR problems as well.

From our implementation of the knapsack problem, we conclude that there are four problems when using Grover's algorithm to solve OR problems: (1) the quadratic speedup ascribed to Grover's algorithm is not obtained because additional repetitions of Grover's algorithm are required, (2) the validity of every solution resulting from a run of Grover's algorithm needs to be verified, (3) since the outcome of Grover's algorithm is probabilistic, there is no guarantee that the optimal solution is found, and (4) Grover's algorithm cannot exploit the structure of a problem. In the literature, several solutions have been proposed to resolve this last issue (e.g., nested quantum search and quantum dynamic programming), however, these solutions still suffer from the first three problems.

The problems we identified, however, do not imply that quantum computing has no potential. For many problems, Grover's algorithm may still allow a speedup, albeit not quadratic. In addition, quantum algorithms show a lot of promise as a heuristic solution procedure, or for problems where obtaining the optimal solution is less important/critical. Given the enormous interest in quantum computing of governments, academics, and the private sector alike, we can safely say that quantum computing is there to stay, just don't expect it to solve all your OR problems (yet).

Appendix

In what follows, we first discuss an algorithm that can be used to determine the exact number of repetitions required by Grover's algorithm. Next, we present a number of experiments that assess the number of repetitions of Grover's algorithm that are required when solving a knapsack problem.

A. Procedure to find exact number of repetitions of Grover's Algorithm

Below, we outline a procedure that can be used to determine the exact number of iterations required by Grover's algorithm. In a first step of the algorithm, we initialize the current probability amplitude of the regular basis states (α_r), the current probability amplitude of the target basis states (α_t), the current iteration (r), and the current probability to to measure a target basis state (p). Next, in each iteration r, we calculate the average probability amplitude as the weighted sum of the probability amplitudes of the regular and the target basis states. We use the average probability amplitude to update the probability amplitude of the target states (i.e., we reflect α_t about $\bar{\alpha}$), and calculate the new probability to successfully identify a target basis state (p'). If this new probability is smaller than the current probability, the current probability is an extremum, and we return r as the number of required repetitions. Otherwise, if $p' \geq p$, we update the current probability, the probability amplitude of the regular states, and the current iteration. Afterwards, we start a new iteration, and continue this process until the required number of iterations has been identified. If the number of correct entries is unknown, Grover's algorithm can be ran for different

Algorithm 2: Procedure to determine the number of repetitions required by Grover's algorithm

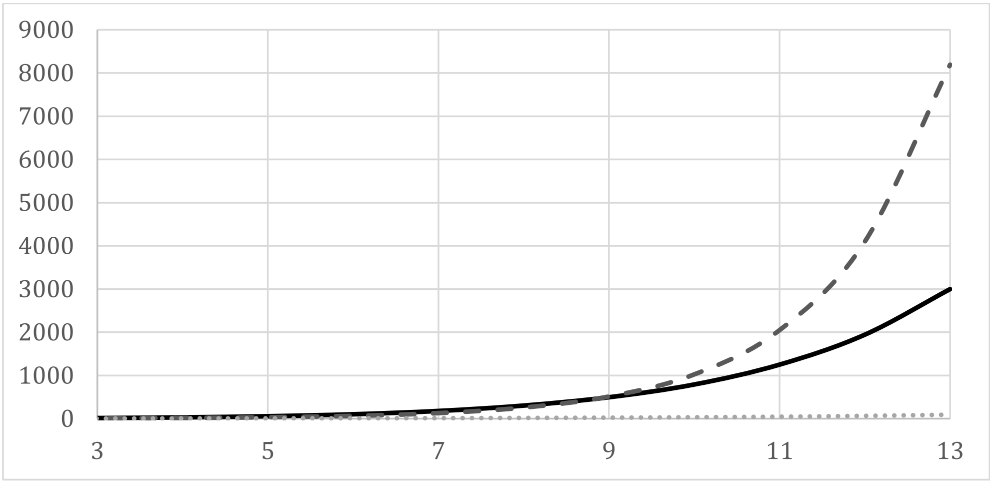
```
Input $n$,$m$;
Initialize $\alpha_r = (\sqrt{2^n})^{-1}$, $\alpha_t = \alpha_r$, $r = 0$, and $p = m2^{-n}$;
do
    $\alpha_t = -\alpha_t$;
    $\bar{\alpha} = 2^{-n}((2^n - m)\alpha_r + m\alpha_t)$;
    $\alpha_t = 2\bar{\alpha} - \alpha_t$;
    $p' = m\alpha_t^2$;
    if $p' < p$ then
        Return $r$;
    $p = p'$;
    $\alpha_r = 2\bar{\alpha} - \alpha_r$;
    $r = r + 1$;
while true;
```

values of m until a matching entry has been found. One approach is to start with $m = 2^n$, and to use divide m by two in a subsequent run if no matching entry was found.

B. Additional experiments to assess the number of repetitions of Grover's algorithm required to solve a knapsack problem

In order to verify the required number of repetitions of Grover's algorithm for various values of n, we perform two additional experiments. In the first additional experiment, we use Algorithm 1 to solve a knapsack problem that has a pool of n items, and assume there is only a single valid solution

Figure 12. Expected number of repetitions of Grover's algorithm (vertical axis) when solving a knapsack problem with n items using Algorithm 1 (black full line) when there is only one valid solution with value 1. For reference, function $f(n) = 2^n$ (dark grey dashed line) and $f(n) = \sqrt{2^n}$ (light grey dotted line) have also been added.



that has value 1 (i.e., only if we evaluate a value $V <= 1$, we are able to find a valid solution). At the start of the algorithm, we let $V_{\min} = 0$ and $V_{\max} = 2^n$. The results of this experiment are presented in Figure 12. In a second additional experiment, we replicate the setting of the first additional experiment, however, we assume there is only a single valid solution that has value 2^n (i.e., we are always able to find a valid solution, no matter the value for V). The results of this experiment are presented in Figure 13. Both experiments show similar results as those presented in Section 6. We do observe that the number of required repetitions of Grover's algorithm is smaller if a solution can always be found. This indicates that less repetitions of Grover's algorithm may be required to solve problems that have many valid solutions that are close to the optimal solution.

References

Aaronson, S. 2008. The Limits of Quantum Computers . *SciAm* **298**(3), 62–69.

Bellman, R. 1962. Dynamic Programming Treatment of the Travelling Salesman Problem. *J. ACM* **9**(1), 61–63.

Benioff, P. 1980. The computer as a physical system: A microscopic quantum mechanical Hamiltonian model of computers as represented by Turing machines. *J. Stat. Phys.*, **22**(5), 563–591.

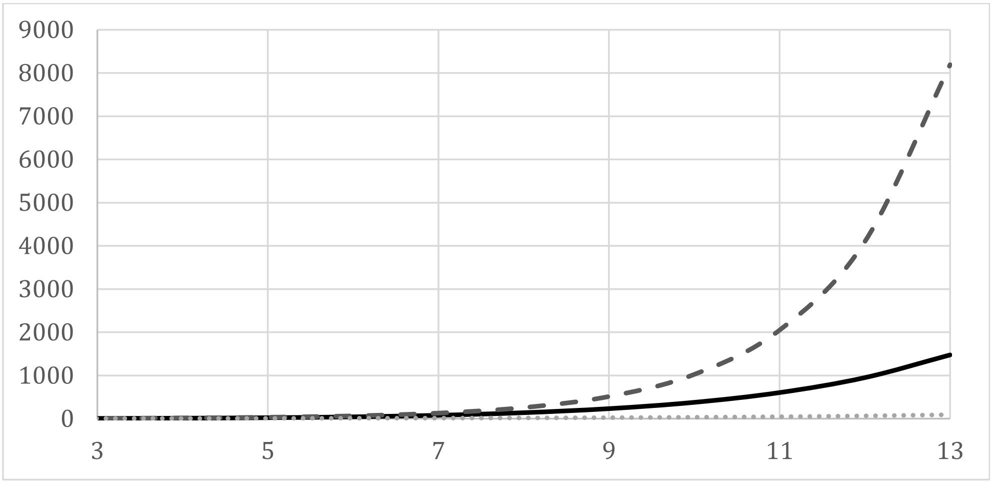
Bennett, C. H., Bernstein, E., Brassard, G., and Vazirani, U. 1997. Strengths and Weaknesses of Quantum Computing. *SIAM J. Comput.* **26**(5) 1510-1523.

Born, M. 1926. Zur Quantenmechanik der Stoßvorgänge. *Z. Phys.* **37**(12), 863–867.

Cerf, N. J., Grover, L. K., and Williams, C. P. 2000. Nested quantum search and structured problems. *Phys. Rev. A* **61**(3), 1–14.

Cleve, R., Ekert, A., Macchiavello, C., and Mosca, M. 1998. Quantum algorithms revisited. *Proc. R. Soc. Lond. A* **454**(1969), 339–354.

Figure 13. Expected number of repetitions of Grover's algorithm (vertical axis) when solving a knapsack problem with n items using Algorithm 1 (black full line) when there is only one valid solution with value 2^n. For reference, function $f(n) = 2^n$ (dark grey dashed line) and $f(n) = \sqrt{2^n}$ (light grey dotted line) have also been added.



Deutsch, D. 1985. Quantum theory, the Church–Turing principle and the universal quantum computer. *Proc. R. Soc. Lond. A* **400**(1818), 97–117.

Deutsch, D., Jozsa, R. 1992. Rapid solution of problems by quantum computation. *Proc. R. Soc. Lond. A***439**(1907), 553—558.

Dirac, P. A. M. 1939. A new notation for quantum mechanics. *Math. Proc. Camb. Philos. Soc.*, **35**(3), 416–418.

Draper, T. G. 2000. Addition on a Quantum Computer. unpublished Preprint at https://arxiv.org/abs/quant-ph/0008033.

Farhi, E., Goldstone, J., and Gutmann, S. 2014. A Quantum Approximate Optimization Algorithm unpublished Preprint at https://arxiv.org/abs/1411.4028.

Feynman, R. P. 1982. Simulating physics with computers. *Int. J. Theor. Phys.*,**21**(6-7), 467–488 .

Grover, L. K. 1996. A fast quantum mechanical algorithm for database search. *Proc. Annu. ACM Symp. Theory Comput.*, 212–219.

Held, M., Karp, R. M. 1962. A Dynamic Programming Approach to Sequencing Problems. *SIAM J. Appl. Math.*,**10**(1), 196–210.

Kellerer, H., Pferschy, U., and Pisinger, D. 2004 *Knapsack Problems.* Springer Berlin Heidelberg, Berlin, Heidelberg.

McKinsey 2021 *Quantum computing: An emerging ecosystem and industry use cases.* McKinsey.

Preskill, J. 2018. Quantum Computing in the NISQ era and beyond. *Quantum 2* **2**, 79–99.

Shor, P.W. 1994 Algorithms for quantum computation: discrete logarithms and factoring. *Proc. Annu. Symp. FOCS*, 124–134.

Xia, H., Li, H., Zhang, H., Liang, Y., and Xin, J. 2018 An Efficient Design of Reversible Multi-Bit Quantum Comparator Via Only a Single Ancillary Bit. *Int. J. Theor. Phys.*, **57**(12), 3727–3744.