EPJ Quantum Technology
a SpringerOpen Journal

RESEARCH Open Access

A quantum circuit to generate random numbers within a specific interval

Francisco Orts[1*], Ernestas Filatovas[2], Ester M. Garzón[1] and Gloria Ortega[1]

*Correspondence:
francisco.orts@ual.es
[1]Informatics Department, University of Almería, Almería, Spain
Full list of author information is available at the end of the article

Abstract

Random numbers are of vital importance in fields such as cyptography and scientific simulations. However, it is well known how difficult it is for classical computers to generate random numbers. This is not the case for quantum computers, which are able to genuinely generate random numbers thanks to the property of superposition and their counter-intuitive concept of measurement. However, despite the simplicity of designing a circuit that generates a random number between 0 and $2^N - 1$ (being N the number of available qubits), designing a quantum circuit to generate a number within a specific interval is far from trivial. This paper proposes a customizable circuit design to generate random numbers. The circuit is non- hardware dependent, it allows fault-tolerance, and it can be used by current quantum devices. Therefore, it is a valuable tool for all those quantum applications and algorithms that need to work with random numbers. Moreover, a comparator circuit has also been designed as part of this work. This comparator is the best currently available in the literature in terms of qubits, T-count, and T-depth. It is therefore a useful tool for any other circuit or algorithm where this operation is needed.

Keywords: Quantum random numbers; Random number generation; Quantum computing; Quantum circuit; Quantum comparator; Clifford + T gates

1 Introduction

Random numbers have a wide variety of applications such as cryptography [1], numerical modelling [2], blockchain [3], or cybersecurity [4]. The quality of random numbers is of vital importance for such applications. For instance, in the case of cryptography, using predictable random numbers would have serious security implications [5]. Generating random numbers using a classical computer is a deterministic process, which is theoretically predictable [6]. Even the modern lattice or hashed-based quantum-safe cryptography methods and algorithms are unable to genuinely generate random numbers [7]. These pseudo-random numbers work well for simple applications, but may be insufficient when higher quality is needed as in the above example of cryptography. It is precisely for this reason that there is great interest in finding alternative ways to generate numbers that are truly random, unpredictable, and secure [8–10].

Quantum computing has emerged as a successful alternative for generating random numbers. In theory of quantum physics, the outcome of most phenomena is not deterministic. Only probabilities of the different possible results can be given. Quantum computing, in its role of emulating quantum physics, brings with it this unpredictability [11]. Properties such as the superposition of quantum states, entanglement, the collapse of states when measured, and the other counter-intuitive properties of quantum physics play a fundamental role in the generation of random numbers using quantum computation [12]. There are a wide range of works in the literature that successfully use random numbers generated by quantum computers [13–15].

Currently, quantum computers are in the so-called NISQ (Noisy Intermediate-Scale Quantum) era. These computers are characterised by a limited amount of resources. Moreover, they are extremely sensitive to noise [16]. There are several options for programming a quantum computer, being the most widespread the use of reversible circuits. Of course, any circuit design must face the aforementioned scarcity of resources. At the same time, they must offer some kind of resistance to errors caused by noise. Therefore, it is important to look for ingenious solutions to reduce the number of involved qubits and quantum gates, but without neglecting the detection and correction of errors [17].

There are several current strategies to reduce the effects of noise [18]. One widespread strategy is to build circuits using only quantum gates belonging to the so-called Clifford + T group [19–22]. Circuits built only with this type of gates can benefit from the use of proven error detection and correction codes. However, among the gates belonging to the Clifford + T group, it is necessary to highlight one of them, the T gate. This gate has a cost in the order of 100 times higher than the cost of the other gates [23]. Its cost is so high that it makes the cost of the other gates almost irrelevant. So, in the literature the number of T-gates is used to measure the resources of quantum circuits [24–26]. The so-called T-count parameter precisely measures the number of T-gates in a circuit. A second metric, the T-depth, indicates the number of T-gates a circuit has on its critical path, allowing an estimation of the depth of the circuit. Keeping the T-count and T-depth values low, as well as involving as few qubits as possible, are probably the most important priorities for building a quantum circuit today [19].

This paper proposes a circuit to generate random numbers in quantum computers and simulators of the NISQ era. The circuit is focused on generating numbers within an interval of possible values to be defined by the user. The aim has been to design an easy to use circuit that generates high-quality random numbers, without having to resort to low-level solutions that may be too challenging or simply beyond the reach of the general public [27–29]. It have also been sought to achieve a circuit with noise tolerance, as well as optimising the number of involved qubits and T gates to allow its viability on NISQ devices.

This paper is structured as follows. Section 2 introduces the formulation and ideas behind the quantum random number generator, and also its limitations. Section 3 presents the developed circuit, indicating how to reproduce it and showing the involved costs. Section 4 discusses the results achieved by the generator. We conclude in Sect. 5.

2 Methods

There are many quantum random number generators in the literature. These generators are of different natures: based on radioactive sources [30], electronic noise [31], atomic systems [32], etc. The proposed circuit is based on the measurement of qubits in superposition, which is the most easy and natural way of generating numbers in quantum computing. This kind of generator uses the collapse that occurs when measuring qubits to obtain random numbers [12].

A random number generator circuit can be designed in a quantum computer using only Hadamard gates. Starting with N qubits initialised in the state $|0\rangle$, it is enough to apply a Hadamard gate on each qubit to put them in a superposition state and then to measure all the qubits. This simple operation, requiring only N Hadamard gates, will produce a random string of N binary values (interpreting the $|0\rangle$ state as 0 and the $|1\rangle$ state as 1). If this string is numerically interpreted, the result is a number in binary notation whose value will always be between 0 and $2^N - 1$. Increasing or decreasing the number of qubits allows us to extend or decrement the upper bound of the interval.

As easy as it is to create a number generator in the interval $[0, 2^N - 1]$, it is not easy to generate numbers in a customized interval. For instance, to generate numbers in the interval $[0, 5]$ to simulate the six possible outcomes of a die can be challenging. In order to represent the largest number, 5, three digits are needed. Using three qubits and the above-mentioned procedure, numbers in the interval $[0, 7]$ can be generated. However, this interval includes several values (6 and 7) outside the range of interest. This problem can be solved by manipulating the obtained result using classical computation. For instance, the value can be checked and, in case an invalid value is obtained, to re-run the circuit until a value in the desired interval was achieved. However, this procedure requires to use classical computation as an intermediary, and may not be a valid (or simply not optimal) solution if we are interesting in using the generator as part of a larger quantum circuit that it does not interfere with a classical computer at that point.

Continuing with the example of generating numbers in the interval $[0, 5]$, another solution could be to add an ancilla qubit and perform some extra operations. Let A_2, A_1, A_0 be the original three qubits, representing the digits of the generated number, and Q being the auxiliary qubit. A_2, A_1, A_0 are set in superposition and measured. Then, the operations $A_2A_1 \oplus Q$ and $Q \oplus A_2$ are sequentially performed. Using this circuit, random numbers in the desired interval are obtained. However, this solution also has drawbacks. First, it is far from generating the numbers uniformly. Secondly, and more importantly, this circuit only works for this specific interval. Customizing the circuit for larger interval will require more ingenuity. Furthermore, for quite large intervals this solution will become unfeasible.

A suitable method for generating random numbers that lie in a subinterval within the interval $[0, 2^N - 1]$ is to modify the probabilistic amplitudes of those states corresponding to values within the subinterval. If the probabilistic amplitudes of the unwanted states cancel out, and those of the wanted states remain uniform, a quality random number generator would be achieved. This manipulation of the amplitudes can be done with Grover's algorithm. Grover's algorithm is shown schematically in Fig. 1. The algorithm (1) sets the

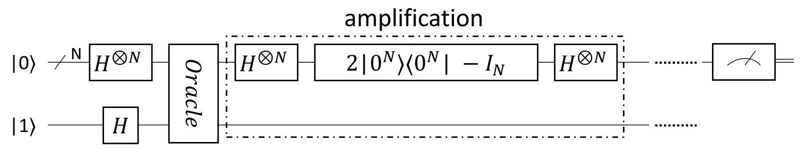

Figure 1 Implementation of Grover's algorithm. Implementation of the Grover's algorithm. It has three parts: a) the initialization, which sets the input into superposition, b) the oracle, and c) the diffusion operator (labelled as amplification). The oracle and the diffusion operator should be repeated a certain number of times in order to increment the probability of the desired result

N qubits in superposition, (2) recognises the solutions to the problem (the numbers of the desired subinterval in this case) through an oracle, and (3) amplifies the amplitudes of the solutions recognised by the oracle. Steps 1 and 2 are always the same and do not allow for customisation beyond how to implement the circuits that perform the operations. However, the oracle is delivered as a black box that the researcher must customise to the problem at hand. The oracle have to add a negative phase to any solution state. Therefore, being w a valid solution and $|x\rangle$ any possible state, the oracle must perform:

$$U_w|x\rangle = \begin{cases} |x\rangle & \text{if } x \neq w, \\ -|x\rangle & \text{if } x = w. \end{cases} \tag{1}$$

Therefore, in order to apply Grover, a valid oracle must be specified that allows a value to be selected within a given range.

2.1 The oracle

In order to amplify the amplitudes of the states corresponding to the numbers within the desired interval $[0, B]$, being B a natural number such that $B > 0$, we need to recognise whether a state represents a number that is less than or equal to B, or not. Working with natural numbers, there is no need to check the lower limit of the interval because a negative number cannot be obtained. Then, it is only necessary to make sure that the generated number A is less than or equal to B. To do this, a half-comparator can be used. This type of circuit compares two numbers A and B and returns 0 if $A \leq B$ and 1 otherwise. There are a wide variety of half-comparators available in the literature [33–35]. However, in this paper we offer a customised comparator that allows us to reduce costs in terms of T gates and number of qubits.

Our comparator is based on the methodology of an adder proposed by Gidney in 2018 [36]. The idea is to use this adder to perform the operation $A - B$ computing $\overline{\overline{A} + B}$. Once the operation is carried out, the sign of the subtraction will contain the result of the comparison. Since we are only interested in the sign, several operations can be simplified or even omitted as they are dedicated to performing the remaining operations of the sum. Moreover, we will not introduce B through the qubits, but we will progressively introduce it through quantum gates depending on whether its digits are 0 or 1. This technique of encoding an integrated number in the circuit itself is not new, and has been used successfully to simplify circuits and optimise resources [37]. The comparator has been built avoiding the use of Toffoli gates when possible, given the high cost of T gates. Instead, temporary logical-AND gates [36] have been used. The most efficient implementation of the Toffoli gate contains 7 T gates, while the temporary logical-AND only involves 4 of these gates. The construction of the proposed comparator and its metrics are detailed in the next section.

2.2 Limitations of the proposed circuit

Theoretically, a quantum computer produces perfectly random numbers. However, it has already been mentioned that current devices are exposed to internal and external noise. The result of this exposure is that current quantum computers do not generate these random numbers uniformly. There are several protocols in the literature, such as those presented by Combarro et al. [11], to improve the uniformity of random number generation by quantum computing. We have left as future work to incorporate this kind of techniques to our circuit to improve the quality of the obtained results.

On the other hand, each current quantum computer has its own internal architecture. This architecture does not usually establish connectivity between all the physical qubits of the machine; instead, each qubit is connected to some but not all of the others. When implementing a circuit from a design, each qubit in the circuit must be mapped to a physical qubit in the machine. All involved qubits in the same operation (for instance, the three qubits acting on a Toffoli gate) must be adjacent. If they are not, it is necessary to apply SWAP gates to move the values to be in adjacent qubits (and reverse the movements once the operation is performed). This causes an obvious increase in the number of quantum gates. Finding the optimal way to minimize the number of SWAP gates, as well as finding the most suitable initial mapping, are subjects of intense study by the research community [38]. It is not the aim of this work to present a customized implementation for each existing quantum device, so neither the extra costs due to SWAP operations nor an estimation of the best initial mapping have been included when analysing the proposed circuit.

3 Results

Since a comparator is required for the construction of the random number generator, the details concerning the comparator will be explained first. Once the comparator has been introduced, the implementation of the generator will be explained.

3.1 Proposed comparator

The circuit compares two N-digit natural binary numbers $a = a_{N-1}\dots a_0$ and $b = b_{N-1}\dots b_0$, with the digits in position $N-1$ being the most significant and the digits in position 0 the least significant. It can be easily built for any digit size $N > 2$ following these steps:

1 For $i = 0$ to $i = N-1$, to apply a Pauli-X gate at every bit a_i to perform $\overline{a}$. These operations can be computed in parallel.

2 b_0 must be introduced in the first ancilla qubit using a Pauli-X gate or an identity gate if $b_0 = 1$ or $b_0 = 0$, respectively.

3 To perform the operation $\overline{a_0}b_0$ using a temporary logical-AND gate instead of a Toffoli gate to save T-count and T-depth. Each temporary logical-AND will require an extra qubit, which must be initialized to the state $\frac{1}{\sqrt{2}}(|0\rangle + e^{\frac{i\pi}{4}}|1\rangle)$.

4 b_0 is uncomputed applying (again, in the first ancilla qubit) a Pauli-X gate or an identity gate if $b_0 = 1$ or $b_0 = 0$, respectively.

5 The next step introduces a loop that will repeat from $i = 1$ to $i = N-1$. Prior to each instruction in this loop, we must codify b_i in a similar way to how b_0 was codified. That is, a Pauli-X gate will be applied to the first ancilla qubit if b_i is 1, or an identity gate will be applied otherwise. Likewise, this action must be repeated after each instruction in the loop to uncompute the value b_i and to prepare the qubit to codify b_{i+1} in the next instruction.

6 For $i = 1$ to $i = N-1$, apply two CNOT gates to compute $(\overline{a_{i-1}}b_{i-1}) \oplus \overline{a_i}$ and $(\overline{a_{i-1}}b_{i-1}) \oplus b_i$. Then, to apply a temporary logical-AND to compute $\overline{a_i}b_i$. A CNOT performing again $(\overline{a_{i-1}}b_{i-1}) \oplus b_i$ is applied to uncompute the previous CNOT. Finally, to apply another CNOT gate to perform $(\overline{a_{i-1}}b_{i-1}) \oplus (\overline{a_i}b_i)$. Each step of the loop must be computed sequentially.

7 The result is given by the last operation of the last iteration computed in the previous step.

The obtained comparator is fully functional, and will return 0 if $a < b$, and 1 otherwise. However, this circuit contains $N - 1$ garbage outputs, which correspond to the auxiliary qubits used to implement the temporary logical-AND gates, except for the last one, which contains the result of the comparison. To reverse the garbage outputs we resort to the Bennett's garbage removal scheme [39] in combination with the measure-and-fixup approach used by the temporary logical-AND gate [36, 40]:

8 For $i = N - 2$ to $i = 1$, to apply a CNOT gate to perform $(\overline{a_{i-1}}b_{i-1}) \oplus (\overline{a_i}b_i)$. A CNOT performing $(\overline{a_{i-1}}b_{i-1}) \oplus b_i$ is applied. Then, to apply an uncomputation gate of the temporary logical-AND to uncompute $\overline{a_i}b_i$. Finally, to apply two CNOT gates to compute $(\overline{a_{i-1}}b_{i-1}) \oplus b_i$ and $(\overline{a_{i-1}}b_{i-1}) \oplus \overline{a_i}$. Again, each step of the loop must be computed sequentially codifying and uncomputing $b_{N-2}, b_{N-1}, \dots b_1$.

9 Uncompute the operation $\overline{a_0}b_0$ using an uncomputation gate of the temporary logical-AND gate. b_0 must be codified at the beginning and uncomputed at the end.

10 Finally, for $i = 0$ to $i = N - 1$ apply a Pauli-X gate at every bit $\overline{a_i}$ to uncompute them. All the qubits except the one that contains the result have been uncomputed.

A complete example for the $N = 4$ case is shown in Fig. 2. In this example, the encoding of b is shown schematically. Each box labelled with a natural number between 0 and $N - 1$ corresponds to a digit of b, so that the i-th box will be an Identity gate (it may be omitted) if $b_i = 0$, or will be a Pauli-X gate if $b_i = 1$.

The logic on which the circuit is based is the one explained in the previous Section: the sign of the operation $\overline{\overline{a} + b}$ is obtained and returned as the result of the comparison between a and b. Since the operation performed is a simplified addition, the circuit construction is started from the least significant digits and moves towards the most significant ones. The auxiliary qubits are used both to reduce the number of T gates and to store the carry of the sum of the previous digits. For instance, the first qubit marked as A in Fig. 2 stores the carry generated from $\overline{a_0}$ and b_0, and so on with the rest of the ancilla qubits for the following pairs $\overline{a_i}$ and b_i. Only the information necessary to generate the carries is stored, while the rest of the operations that in a normal adder would allow calculating the rest of the result of the sum are discarded.

For the general case of N digits, N qubits are needed to encode a. $N + 1$ auxiliary qubits are also used to perform auxiliary operations, making a total of $2N + 1$ qubits. N Pauli-X gates to invert a_i values, and $2N$ to encode b (depending on the 1 number of the b

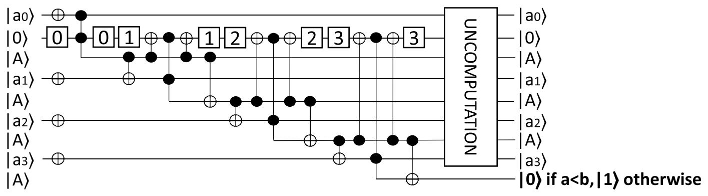

Figure 2 Proposed comparator. Scheme of the proposed comparator, for the $N = 4$ digit case. Two numbers a and b are compared. a_i represent the digits of a, whereas the digits of b are encoded in the circuit using quantum gates. Boxes labelled with numbers $0 \dots N - 1$ represent such digits, so that boxes marked as 0 will be Pauli-I or Pauli-X gates depending on whether b_0 is 0 or 1, respectively; boxes marked as 1 will be Pauli-I or Pauli-X gates depending on whether b_1 is 0 or 1, respectively, and so on for each digit of b. Qubits labelled with an A are prepared in the $\frac{1}{\sqrt{2}}(|0\rangle + e^{\frac{i\pi}{4}}|1\rangle)$ state for be used with temporary logical-AND gates

expression), add up to a maximum of $3N$ Pauli-X gates. $4(N-1)$ CNOT gates, and N temporary logical-AND gates are also involved in the whole process. On the other hand, reversing the garbage outputs will require N Pauli-X gates for a_i values and a maximum of $2(N-1)$ gates for b (making a maximum of $3N-2$ Pauli-X gates), $4N-7$ CNOT gates, and $N-1$ gates for uncompute the temporary logical-AND operations. The total numbers of the circuit are:

- Number of qubits: $2N+1$
- Pauli-X gate: $6N-2$
- CNOT gate: $8N-11$
- Temporary logical-AND gate: N
- Uncomputation gate for the temporary logical-AND operation: $N-1$

From the mentioned gates, only the temporary logical-AND gate involves T gates. Since the circuit contains N temporary logical-AND gates, and that the T-count of each temporary logical-AND gate is 4, it can be established that the T-count of the whole circuit is $4N$. Moreover, since the N temporary logical-AND gates are executed sequentially, and since their T-depth is 2, it can be established that the T-depth of the comparator is $2N$.

3.2 Quantum random number generator

The necessary steps to build the random number generator circuit for N-digit numbers in the half-open interval $[0, B)$, assuming that B can be represented by N digits, are as follows:

1. To design a comparator circuit using the methodology explained in the previous subsection, coding $a = 0$ (i.e., keeping all inputs $a_i = 0$) and $b = B$ (using the corresponding CNOT and Identity gates). It must only be designed until the result is obtained, i.e., without applying the uncomputation part yet. This circuit will henceforth be referred to as *Comparator*.
2. To obtain the inverse circuit of the previous one. In the previous subsection we explained how to reverse the garbage outputs of the comparator, which is a different process. The circuit that must be obtained now must also reverse the output of the comparator itself. That is to say, it must be identical to *Comparator*, but in reverse. We will call this second circuit *Uncomputation of the comparator*.
3. To prepare N qubits in the $|0\rangle$ state, and one qubit in the $|1\rangle$ state. We apply a Hadamard gate to each of these qubits. At this point, we will have N qubits in the $|+\rangle$ state, and one qubit in the $|-\rangle$ state.
4. The Oracle phase:
   (a) To implement *Comparator*, so that the N qubits in the $|+\rangle$ state match the a_i inputs of *Comparator*.
   (b) After *Comparator*, to apply a CNOT gate. The control qubit will be the qubit resulting from *Comparator*, and the target qubit will be the qubit we initialise to the $|-\rangle$ state.
   (c) To apply *Uncomputation of the comparator* to reverse *Comparator*.
5. Amplification phase:
   (a) To apply a Hadamard gate to each of the N qubits that were initially set to the $|+\rangle$ state. After the Hadamard gates, to apply a Pauli-X to each of these qubits.
   (b) Of the same N qubits above, to apply only to the most significant one a Hadamard gate. With the remaining $N-1$ qubits acting as control qubits, to apply a controlled Toffoli gate, with the most significant qubit being the target qubit.

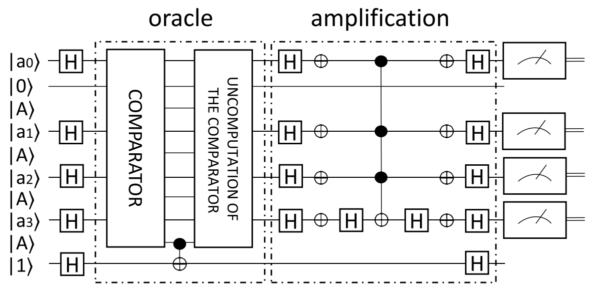

Figure 3 Proposed random number generator. Design of the proposed quantum random number generator, for the $N = 4$ case. Only one more qubit is needed than in the comparator circuit. Three steps can be clearly distinguished: (1) the formation of a uniform superposition of all possible states with 4 qubits using Hadamard transform gates, (2) the oracle, consisting of applying the comparator circuit of Fig. 2, storing the result in the new qubit, and decomputing the circuit, and (3) the application of an amplitude amplification function

(c) To apply a Pauli-X gate to each of the previous N qubits. After the Pauli-X gates, to apply a Hadamard gate to each of these qubits.

6 To repeat steps 4 and 5 $\|\frac{\pi}{4}\sqrt{\frac{2^N}{l}}\|$ times, being l the size of the interval $[0, B)$.

7 To measure the N qubits that was initially set to $|+\rangle$. They will contain the random number.

A scheme of the quantum random number generator circuit, for the case $N = 4$, is shown in Fig. 3. The generator circuit can be easily adapted to generate random numbers in the closed interval $[0, B']$ by simply setting $B' = B + 1$. Likewise, a circuit that generates random numbers in the closed interval $[B, 2^N - 1]$ can be obtained by inverting the output of *Comparator* using a Pauli-X gate (this gate must also be taken into account when designing *Uncomputation of the comparator*). To generate numbers in the half-open interval $(B', 2^N - 1]$, it is only necessary to set $B' = B + 1$.

The circuit involves a total of $2N + 2$ qubits. In terms of quantum gates, the first part involves $N + 1$ Hadamard gates. The Oracle phase involves:

- Pauli-X gates: $6N$.
- CNOT gates: $8N - 7$.
- Temporary logical-AND gates: N.
- Uncomputation gates for the temporary logical-AND operation: N

Finally, the amplification phase involves:

- Hadamard gates: $2N + 3$.
- Pauli-X gates: $2N$.
- N-qubit Toffoli gates: 1.

Bearing in mind that the oracle and the amplification phases can be repeated I times, the final numbers are:

- Number of qubits: $2N + 2$
- Hadamad gates: $(N + 1) + I \times (2N + 3)$
- Pauli-X gate: $I \times 8N$
- CNOT gate: $I \times (8N - 7)$
- Temporary logical-AND gate: $I \times N$
- Uncomputation gate for the temporary logical-AND operation: $I \times (N - 1)$
- N-qubit Toffoli gates: I.

4 Discussion

The main contribution of this work is the random number generator circuit. However, the comparator presented has been designed as part of this work. Therefore, not only the results of the generator will be discussed, but also those obtained with the comparator circuit.

4.1 Discussion on the comparator

The circuits included in the following comparison have been implemented using the Qiskit Programming Language and run in the quantum simulator *simulator_mps*, available in the IBM Quantum platform [41].

Table 1 shows a comparison between the comparator circuit proposed in this work and the most important half-comparators available in the state-of-the-art. This comparison has been carried out in terms of T-count, T-depth, and number of necessary qubits. In view of the results, it can be seen that there is no other circuit that outperforms the proposed comparator in any of the indicated metrics, except for the case of T-depth. In terms of T-count, the proposed circuit has a value of $4N$. This value is matched by the circuit of Orts et al. [35] (labelled as Orts et al.[a] in Table 1). The rest of the circuits have values of more than double the results obtained by these two comparators.

Regarding of the T-depth, the best circuit is a logarithmic depth version of the Orts et al. comparator (presented in the same paper as the previous one, and labelled as Orts et al. [b] in Table 1). However, this second comparator achieves this depth reduction at the cost of an increase in the T-count and, above all, in the number of needed qubits, being in these terms the worst circuit of the whole comparison. This circuit needs $4N - 2W(N) - 2log(N)$ ancilla qubits (being $W(N)$ the number of ones in the binary expansion of N) to perform the comparison between numbers of N qubits (plus $2N$ qubits to encode the numbers). This number is too high for current quantum computers and simulators. For feasibility purposes, it can be stated that the following two circuits with the best T-depth (Orts et al. [a], and the proposed circuit, with a T-depth of $2N$, almost three times less than the rest of the circuits) are the most suitable options.

Finally, in terms of the number of qubits, the best circuits are the proposed circuit and the circuit of Li et al [33], with a value of $2N + 1$. However, the circuit of Li et al. has almost three times the T-count and T-depth of the proposed circuit. The rest of the circuits, except for those presented in Orts et al., involve a single qubit increment but with very high T-count and T-depth values. Furthermore, the Orts et al. circuits, despite reducing these two metrics, require a higher number of qubits.

Previously, the most suitable choice was either the circuit of Li et al. when the number of qubits was to be optimised, or the circuit of Orts et al.[a] when a reduced number of

Table 1 Evaluation of the best comparator circuits in terms of T-count, T-depth and number of qubits as functions of N. $W(N)$ is the number of ones in the binary expansion of N

Circuit Comparator	T-count	T-depth	Number of qubits
Xia et al. (2018) [42]	$14N$	$6N$	$2N + 2$
Xia et al. (2019) [43]	$14N - 7$	$6N - 3$	$2N + 2$
Li et al. (2020) [33]	$14N - 7$	$6N - 3$	$2N + 1$
Xia et al. (2020) [34]	$14N$	$6N$	$2N + 2$
Orts et al. [a] (2021) [35]	$4N$	$2N$	$3N$
Orts et al. [b] (2021) [35]	$12N - 8W(N) - 4Log(N)$	$Log(N)$	$6N - 2W(N) - 2log(N)$
Proposed comparator	$4N$	$2N$	$2N + 1$

T-gates was required. The circuit proposed in this work maintains the same number of qubits than the Li et al. circuit, but keeping the optimal levels of T gates presented by the Orts et al. circuit. It is, therefore, the most optimised comparator in such metrics currently available to the authors' knowledge.

4.2 Discussion on the random number generator circuit

The generator circuit has been implemented using the Qiskit Programming Language and run in the quantum noisy modeling simulator *ibmq_qasm_simulator*, available in the IBM Quantum platform [41], according to what is specified below in this subsection.

The utility of the circuit has been measured in a context where it is useful, as explained below. Given N qubits, the numbers in the interval $[0, 2^N - 1]$ can be represented. Random numbers can be produced in this complete interval using the generator circuit, but this is pointless as it can be done simply by applying N Hadamard gates to N qubits initialised to $|0\rangle$, and then simply measuring them. Likewise, the cases in which there is only one possible value to be produced can also be discarded since such an operation is meaningless. We can also discard the cases where there are only two possible values in the interval (intervals $[0, 1]$ or $[2^N - 2, 2^N - 1]$) as in such cases the least significant qubits could be set to 0 or 1 and randomly decide only the last qubit. The latter could also be set for intervals whose size coincides with a power of 2, but for simplicity such cases are included.

On the other hand, tests have been done for circuits generating numbers between 3 and 8 digits. The largest current quantum computers whose access is not exclusively restricted to the manufacturer have a capacity of around 20 qubits (e.g. IBM Q 20 Tokyo or IBM Q 20 Austin computers). The random number generator requires $2N + 2$ qubits. Considering a 20 qubit computer, the maximum number of digits that can be worked with is $N = 8$. This circuit is focused on current NISQ-era computers and simulators, so larger sizes have not been tested.

Figures 4 and 5 show the number of required circuit iterations (oracle and amplification phases) for each value of N as a function of the size of the solution interval l. Figure 4 displays the number of iterations for the 3, 4, and 5 digit cases (in blue, orange, and green, respectively). In the Figure, the Y-axis represents the number of iterations needed, while

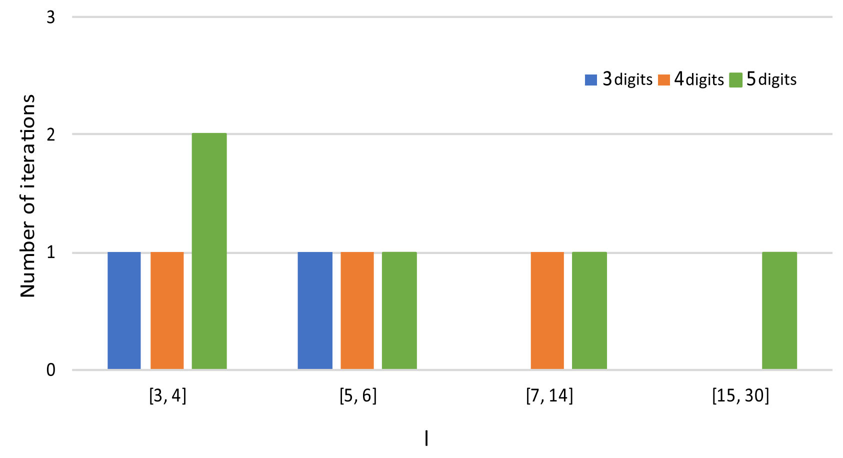

Figure 4 Number of iterations required for small sizes. Number of iterations needed to generate 3, 4, and 5-digit random numbers depending on the possible number of solutions (interval size)

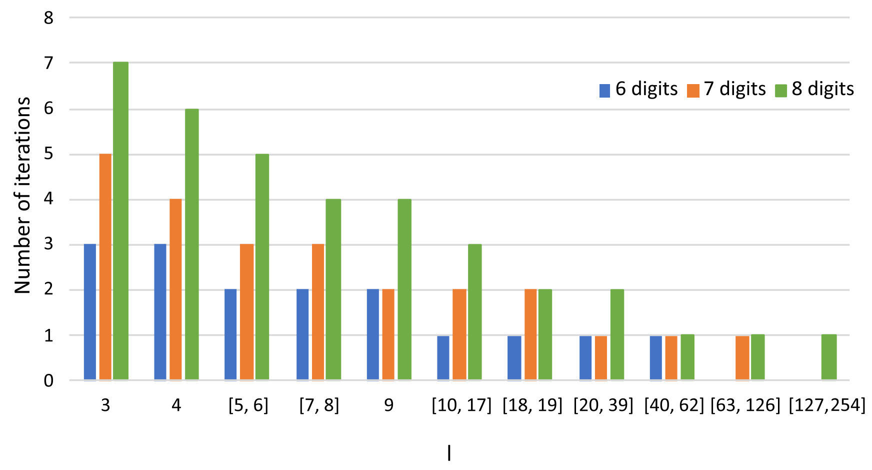

Figure 5 Number of iterations required for big sizes. Number of iterations needed to generate 6, 7, and 8-digit random numbers depending on the possible number of solutions (interval size)

Table 2 Count of N-digit possible numbers and minimum interval width (value and percentage) to compute the circuit using only one iteration. For any value (percentage) higher than indicated here, only a single iteration is required

Number of digits (N)	Distinct N-digit numbers (2^N)	Minimum interval width (l_{min})	Minimum interval width ($l_{min}/2^N \times 100$)
3	8	> 0	–
4	16	> 0	–
5	32	> 4	15.625
6	64	> 9	15.625
7	128	> 19	15.625
8	256	> 39	15.625

the X-axis indicates the size of the interval. For instance, the value [7 – 14] on the X-axis indicates that this includes intervals with between 7 and 14 numbers (both cases included). The minimum number of solutions (the minimal width of the interval, l) addressed is 3 in all cases, as justified above. The maximum number of possible random values in the interval is 6, 14, and 30 for 3 digits, 4 digits, and 5 digits, respectively. For the cases of 3 and 4 digit numbers, only one Grover's iteration is needed in all cases. For the $N = 5$ case, a single iteration is required for 5 or more solutions. 2 iterations are required when there are 3 or 4 possible solutions.

Figure 5 shows the same information as Fig. 4, but for sizes $N = 6, 7$, and 8 (in blue, orange, and green, respectively). The minimum number of solutions is still 3. The maximum number of solutions is 62, 127, and 254, respectively. Each size reaches the optimal number of iterations (i.e., a single iteration) from 10, 20, and 40 iterations, respectively. It takes 2 iterations starting at 5, 9, and 18 solutions, respectively. If these values are reduced, the number of iterations quickly starts to become unfeasible, as shown in Fig. 5.

Table 2 summarises the minimum interval sizes to run the circuit with a single iteration. It also shows this value as a percentage of the number of distinct values that can be obtained using N digits. In the best cases, all possible values results in a resource-optimal circuit. On the other hand, for larger sizes it is necessary to include at least about a 15.625%

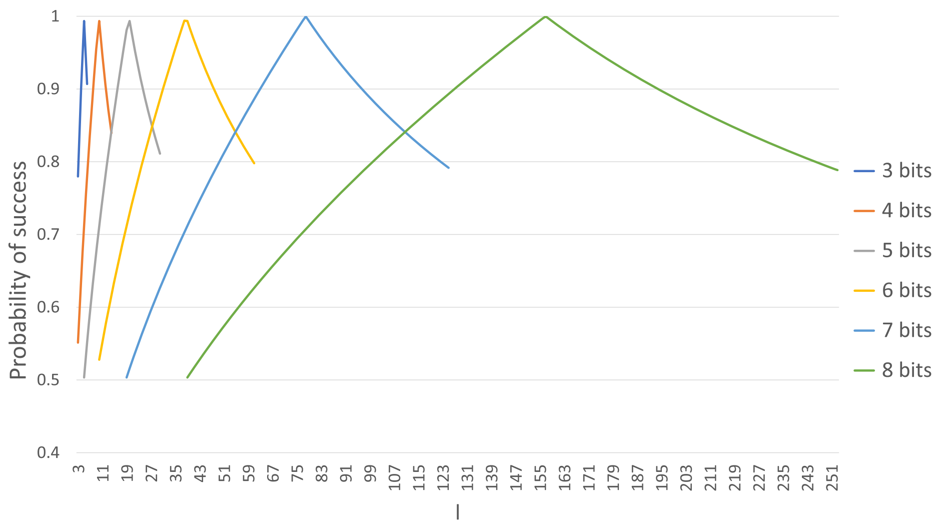

Figure 6 Probabilities of success. Probabilities of success for every interval size (number of solutions *l*) using the different number of qubits

of the possible values if the most efficient circuit possible is required. For these single iteration cases, the metrics are:

- Number of qubits: $2N + 2$
- Hadamad gates: $3N + 4$
- Pauli-X gate: $8N$
- CNOT gate: $8N – 7$
- Temporary logical-AND gate: N
- Uncomputation gate for the temporary logical-AND operation: $N – 1$
- N-qubit Toffoli gates: 1.

For each case, 10 tests (of 1024 runs each) were carried out in order to measure the effectiveness of the circuit. Figure 6 shows the average probabilities of successfully obtaining a value for the number of digits and optimal interval configurations (the cases where the circuit needs only one iteration) listed in Table 2. The shape of the plots shown in Fig. 6 conforms to what is expected by Grover's algorithm [44]. The algorithm ensures probabilities of success above 50%. The probabilities increase until they reach a maximum, at which point they start to decrease. This pattern will continue to repeat if the number of iterations is increased, but each time with a longer repetition period. Values around 50% may be adequate in certain domains, but are insufficient for the problem at hand. However, in Fig. 6 it can be seen that random numbers with a probability always higher than 80% can be obtained for any interval if the right number of qubits is used. For instance, to generate a number in the interval $[0, 76]$ (i.e. $l = 77$) using 8 qubits, will have a probability of success of only 70% according to Fig. 6. However, the probability of success becomes almost 100% if 7 qubits are used instead. We can then define a number of qubits for each interval so that we can ensure that the probabilities never go below 80%, as shown in Fig. 7. We can even define certain interval sizes where the probability is always greater than 90% or even 95%, as shown in Table 3. Moreover, all possible intervals can be covered using a single Grover iteration.

Table 3 Average probability of success of obtaining a number in the desired interval for the different numbers of digits. To calculate the average value, only those intervals of each size for which a single iteration of the circuit is needed have been included. The efficient and optimal intervals columns indicates (for each size) the intervals in which the probability of success is greater than 90% and 95%. For instance, for the case $N = 3$, the efficient interval column indicates that the intervals [0, 4], [0, 5], and [0, 6] produce random numbers in those intervals with a probability of success greater than 90%

Number of digits (N)	Quality (average)	Efficient intervals (probability > 90%)	Optimal intervals (probability > 95%)
3	0.895	[0, 4 – 6]	[0, 5]
4	0.827	[0, 8 – 12]	[0, 9 – 10]
5	0.825	[0, 15 – 24]	[0, 18 – 21]
6	0.824	[0, 32 – 48]	[0, 35 – 42]
7	0.823	[0, 64 – 95]	[0, 71 – 86]
8	0.822	[0, 128 – 194]	[0, 142 – 173]

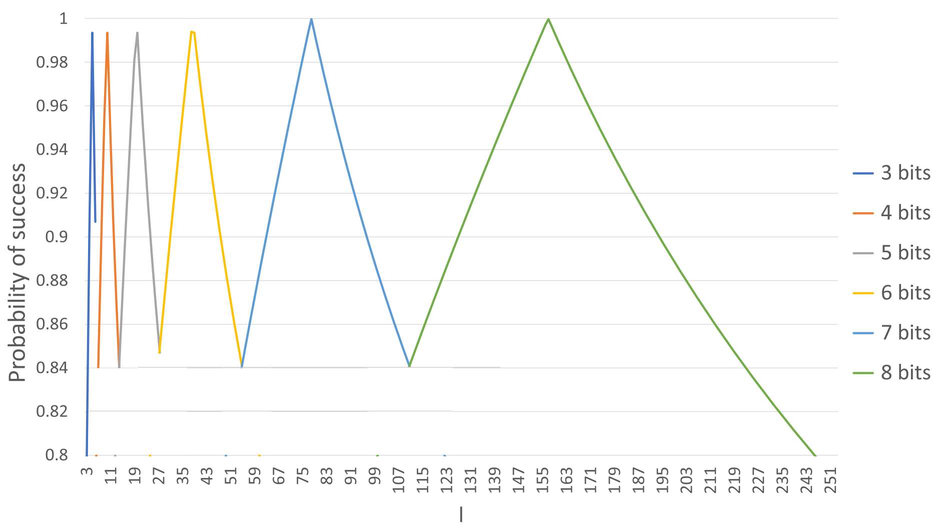

Figure 7 Number of qubits for each interval size. Number of recommended qubits for each possible interval size. Using these number of qubits for the indicated interval, the probabilities of success are maximized

Finally, the uniformity of the numbers has been measured using the TestU01 software [45]. Only the values generated within the interval have been included in the measurement, thus discarding the erroneous values. Table 4 shows the percentage of uniformity achieved for each number of digits as described in Table 3. The quality of the numbers has been tested both without noise and with noise. In the latter case, the configuration described by Combarro et al. [11] was used in order to have some reference value against which to compare the results. In the work of Combarro et al., the Monobit test is used to measure the quality, obtaining a value of 0.9899 in the ideal case without noise, and 0.3962 in the case with noise. The values achieved by our generator are similar to those measured by Combarro et al. The slight improvement of our values can be attributed to the fact that we are discarding in advance the values that fall outside the intervals. Our results confirms what Combarro et al. have already indicated: the numbers generated in the presence of noise using this methodology are not uniform. In case such uniformity is needed, the application of techniques and protocols that improve the quality of the numbers generated must be used. We leave the application of such protocols for future work, as mentioned in previous sections.

Table 4 Measure of uniformity (mean) of numbers generated using *N* qubits without noise and with noise. The TestU01 software has been used for uniformity measurement

Number of digits (*N*)	Uniformity (no noise)	Uniformity (with noise)
3	0.9902	0.5843
4	0.9887	0.5837
5	0.9886	0.5823
6	0.9871	0.5812
7	0.9869	0.5778
8	0.9866	0.5745

5 Conclusions

In this work, a functional circuit has been proposed to generate random numbers in quantum devices. The circuit is fault-tolerant, and it is optimized in terms of number of qubits, T-count, and T-depth. Its design is customisable for a user-specified interval. Detailed steps are given to reproduce the circuit for any digit size and interval configuration. For each of these cases, the required resources are indicated. The effectiveness of the circuit has been also tested. It has been indicated in which cases it is feasible with respect to the number of resources, and its probability of success has been measured. Complementary, we have indicated those cases in which the circuit works exceptionally well, with a probability of success of over 90% or even 95%.

Another important contribution of this work is that a reversible comparator has been presented. This circuit is used as a sub-circuit of the random number generator, but is valid for use in other quantum circuits and algorithms. We have shown that this comparator is the best currently available in the literature in terms of number of qubits, T-count, and T-depth.

Acknowledgements
Not applicable.

Funding
This work has been supported by the projects: RTI2018-095993-B-I00, PID2021-123278OB-I00 (funded by MCIN/AEI/10.13039/501100011033/ FEDER "A way to make Europe"); P20_00748, UAL2020-TIC-A2101, and UAL18-TIC-A020-B (funded by Junta de Andalucía and the European Regional Development Fund, ERDF); and the Research Council of Lithuania (LMTLT), agreement No. S-MIP-21-53.

Availability of data and materials
Data sharing is not applicable to this article as no datasets were generated or analysed during the current study.

Declarations

Competing interests
The authors declare no competing interests.

Author contributions
Francisco Orts conceived the experiment and carried out the technical and experimental work. All authors discussed the results and contributed to the writing of the manuscript. All authors read and approved the final manuscript.

Author details
[1]Informatics Department, University of Almería, Almería, Spain. [2]Institute of Data Science and Digital Technologies, Vilnius University, Vilnius, Lithuania.

Received: 4 July 2022 Accepted: 17 May 2023 Published online: 25 May 2023

References
1. Pirandola S, Andersen UL, Banchi L, Berta M, Bunandar D, Colbeck R, Englund D, Gehring T, Lupo C, Ottaviani C et al. Advances in quantum cryptography. Adv Opt Photonics. 2020;12(4):1012–236.

2. Pan C, Chen G, Tang J, Wu K. Numerical modeling of partial discharges in a solid dielectric-bounded cavity: a review. IEEE Trans Dielectr Electr Insul. 2019;26(3):981–1000.
3. Chatterjee K, Goharshady AK, Pourdamghani A. Probabilistic smart contracts: secure randomness on the blockchain. In: 2019 IEEE international conference on blockchain and cryptocurrency (ICBC). IEEE; 2019. p. 403–12.
4. Cambou B, Telesca D. Ternary computing to strengthen cybersecurity. In: Science and information conference. Berlin: Springer; 2018. p. 898–919.
5. Gehring T, Lupo C, Kordts A, Solar Nikolic D, Jain N, Rydberg T, Pedersen TB, Pirandola S, Andersen UL. Homodyne-based quantum random number generator at 2.9 gbps secure against quantum side-information. Nat Commun. 2021;12(1):1–11.
6. Avesani M, Marangon DG, Vallone G, Villoresi P. Source-device-independent heterodyne-based quantum random number generator at 17 gbps. Nat Commun. 2018;9(1):1–7.
7. Huang L, Zhou H, Feng K, Xie C. Quantum random number cloud platform. npj Quantum Inf. 2021;7(1):1–7.
8. Ryan C, Kshirsagar M, Vaidya G, Cunningham A, Sivaraman R. Design of a cryptographically secure pseudo random number generator with grammatical evolution. Sci Rep. 2022;12(1):1–10.
9. Crocetti L, Di Matteo S, Nannipieri P, Fanucci L, Saponara S. Design and test of an integrated random number generator with all-digital entropy source. Entropy. 2022;24(2):139.
10. Cang S, Kang Z, Wang Z. Pseudo-random number generator based on a generalized conservative sprott-a system. Nonlinear Dyn. 2021;104(1):827–44.
11. Combarro EF, Carminati F, Vallecorsa S, Ranilla J, Rúa IF. On protocols for increasing the uniformity of random bits generated with noisy quantum computers. J Supercomput. 2021;77(8):8063–81.
12. Mannalath V, Mishra S, Pathak A. A comprehensive review of quantum random number generators: concepts, classification and the origin of randomness. 2022. arXiv preprint. arXiv:2203.00261.
13. Bacco D, Vagniluca I, Da Lio B, Biagi N, Della Frera A, Calonico D, Toninelli C, Cataliotti FS, Bellini M, Oxenløwe LK et al. Field trial of a three-state quantum key distribution scheme in the Florence metropolitan area. EPJ Quantum Technol. 2019;6(1):5.
14. Lowndes D, Frick S, Hart A, Rarity J. A low cost, short range quantum key distribution system. EPJ Quantum Technol. 2021;8(1):15.
15. Polnik M, Mazzarella L, Di Carlo M, Oi DK, Riccardi A, Arulselvan A. Scheduling of space to ground quantum key distribution. EPJ Quantum Technol. 2020;7(1):3.
16. Bharti K, Cervera-Lierta A, Kyaw TH, Haug T, Alperin-Lea S, Anand A, Degroote M, Heimonen H, Kottmann JS, Menke T et al. Noisy intermediate-scale quantum algorithms. Rev Mod Phys. 2022;94(1):015004.
17. Preskill J. Quantum computing in the NISQ era and beyond. Quantum. 2018;2:79.
18. Nielsen MA, Chuang IL. Quantum computation and quantum information: 10th anniversary edition. New York: Cambridge University Press; 2011.
19. Thapliyal H, Muñoz-Coreas E, Khalus V. Quantum circuit designs of carry lookahead adder optimized for T-count, T-depth and qubits. Sustain Comput: Inf Syst. 2021;29:100457.
20. Gayathri S, Kumar R, Dhanalakshmi S, Dooly G, Duraibabu DB. T-count optimized quantum circuit designs for single-precision floating-point division. Electronics. 2021;10(6):703.
21. Mosca M, Mukhopadhyay P. A polynomial time and space heuristic algorithm for T-count. Quantum Sci Technol. 2021;7(1):015003.
22. Orts F, Ortega G, Filatovas E, Garzón EM. Implementation of three efficient 4-digit fault-tolerant quantum carry lookahead adders. J Supercomput. 2022;78:13323–41.
23. Abdessaied N, Amy M, Soeken M, Drechsler R. Technology mapping of reversible circuits to Clifford + T quantum circuits. In: 2016 IEEE 46th international symposium on multiple-valued logic (ISMVL). IEEE; 2016. p. 150–5.
24. Kissinger A, van de Wetering J, Vilmart R. Classical simulation of quantum circuits with partial and graphical stabiliser decompositions. 2022. arXiv preprint. arXiv:2202.09202.
25. Fösel T, Niu MY, Marquardt F, Li L. Quantum circuit optimization with deep reinforcement learning. 2021. arXiv preprint. arXiv:2103.07585.
26. de Brugière TG, Baboulin M, Valiron B, Martiel S, Allouche C. Reducing the depth of linear reversible quantum circuits. IEEE Trans Quantum Eng. 2021;2:1–22.
27. Schmidt H. Quantum-mechanical random-number generator. J Appl Phys. 1970;41(2):462–8.
28. Vincent C. The generation of truly random binary numbers. J Phys E, Sci Instrum. 1970;3(8):594.
29. Vincent C. Precautions for accuracy in the generation of truly random binary numbers. J Phys E, Sci Instrum. 1971;4(11):825.
30. Alkassar A, Nicolay T, Rohe M. Obtaining true-random binary numbers from a weak radioactive source. In: International conference on computational science and its applications. Berlin: Springer; 2005. p. 634–46.
31. Gude M. Concept for a high performance random number generator based on physical random phenomena. Frequenz. 1985;39(7–8):187–90.
32. Pironio S, Acín A, Massar S, de La Giroday AB, Matsukevich DN, Maunz P, Olmschenk S, Hayes D, Luo L, Manning TA et al. Random numbers certified by Bell's theorem. Nature. 2010;464(7291):1021–4.
33. Li H-S, Fan P, Xia H-Y, Peng H, Long G-L. Efficient quantum arithmetic operation circuits for quantum image processing. Sci China, Phys Mech Astron. 2020;63:1–13.
34. Xia H-Y, Zhang H, Song S-X, Li H, Zhou Y-J, Chen X. Design and simulation of quantum image binarization using quantum comparator. Mod Phys Lett A. 2020;35(9):2050049.
35. Orts F, Ortega G, Cucura A, Filatovas E, Garzón E. Optimal fault-tolerant quantum comparators for image binarization. J Supercomput. 2021;77(8):8433–44.
36. Gidney C. Halving the cost of quantum addition. Quantum. 2018;2:74.
37. Pérez-Salinas A, Cervera-Lierta A, Gil-Fuster E, Latorre JI. Data re-uploading for a universal quantum classifier. Quantum. 2020;4:226.
38. Bhattacharjee D, Saki AA, Alam M, Chattopadhyay A, Ghosh S. Muqut: multi-constraint quantum circuit mapping on NISQ computers. In: 2019 IEEE/ACM international conference on computer-aided design (ICCAD). IEEE; 2019. p. 1–7.
39. Bennett CH. Logical reversibility of computation. IBM J Res Dev. 1973;17(6):525–32.

40. Jones C. Low-overhead constructions for the fault-tolerant Toffoli gate. Phys Rev A. 2013;87(2):022328.
41. Qiskit 0.36.2 documentation. https://qiskit.org/documentation/. Accessed: 2022-05-31
42. Xia H-Y, Li H, Zhang H, Liang Y, Xin J. An efficient design of reversible multi-bit quantum comparator via only a single ancillary bit. Int J Theor Phys. 2018;57(12):3727–44.
43. Xia H-Y, Li H, Zhang H, Liang Y, Xin J. Novel multi-bit quantum comparators and their application in image binarization. Quantum Inf Process. 2019;18(7):229.
44. Cafaro C, Mancini S. On Grover's search algorithm from a quantum information geometry viewpoint. Phys A, Stat Mech Appl. 2012;391(4):1610–25.
45. L'ecuyer P, Simard R. Testu01: ac library for empirical testing of random number generators. ACM Trans Math Softw. 2007;33(4):1–40.

Publisher's Note