Genetic Synthesis of Compact Quaternary Reversible Comparators for Quantum Computing

ASMA TAHERI MONFARED, University of Bergamo, Bergamo, Italy
ANDREA BOMBARDA, University of Bergamo, Bergamo, Italy
ANGELO GARGANTINI, University of Bergamo, Bergamo, Italy
VALENTINA CIRIANI, Department of Computer Science, Università degli Studi di Milano, Milan, Italy

Reversible logic is fundamental to quantum circuit design, as quantum operations are inherently information-preserving and reversible . While most quantum synthesis methods rely on binary logic, quaternary reversible computing can increase data density, reduce circuit width, and potentially lead to more efficient realizations. We introduce a genetic-algorithm-based approach for designing compact quaternary reversible comparator circuits, which are important components in quantum architectures. This technique utilizes a gate library based on extended Shift and Muthukrishnan–Stroud gates tailored to quaternary systems. Chromosomes encode sequences of quaternary gates, and evolutionary operators search for configurations with minimal quantum cost. Although demonstrated on comparator circuits, the approach applies to any quaternary reversible circuit defined by its truth table. The synthesis process occurs in two phases: candidate circuits first evolve toward correct behavior; then correct circuits are optimized to obtain compact implementations. We evaluate the approach on comparators performing lower-than, greater-than, and equality operations, as well as on a 1-qudit full comparator. The method achieves average quantum cost improvements of about 30% for restoring and 58% for non-restoring configurations compared to existing designs. These reductions support more efficient and more error-resilient quantum circuits, showing that this approach is a strong candidate for quaternary quantum systems.

CCS Concepts: • **Theory of computation → Logic**; • **Hardware → Reversible logic**; **Quantum technologies**; • **Computing methodologies**;

Additional Key Words and Phrases: Quaternary logic, reversible computing, comparator circuits, genetic algorithm, quantum circuit synthesis, circuit optimization

1 Introduction

Recently, the minimization of power dissipation has become a critical concern in the design of integrated circuits, particularly for digital systems. Indeed, conventional irreversible logic leads to energy loss due to information erasure, as established by Landauer [42]. Bennett later showed that reversible computation can, in principle, avoid this fundamental energy dissipation [4]. As a result, reversible logic has attracted substantial interest in recent years, as it provides a pathway toward low-power computing and energy-efficient hardware design [88].

Reversible logic gates are defined as n-input, n-output circuits with a one-to-one correspondence between input and output vectors [70, 71, 89].This property ensures that the input can be uniquely reconstructed from the output. Unlike classical circuits, reversible circuits do not allow fan-out or feedback [65, 70]. Reversible logic forms the basis of low-power computing, particularly in quantum technologies [40, 65, 70]. Previous studies

Authors' Contact Information: Asma Taheri Monfared, University of Bergamo, Bergamo, Lombardy, Italy; e-mail: asma.taherimonfared@unibg.it; Andrea Bombarda, University of Bergamo, Bergamo, Lombardy, Italy; e-mail: andrea.bombarda@unibg.it; Angelo Gargantini, University of Bergamo, Bergamo, Lombardy, Italy; e-mail: angelo.gargantini@unibg.it; Valentina Ciriani, Department of Computer Science, Università degli Studi di Milano, Milan, Lombardy, Italy; e-mail: valentina.ciriani@unimi.it.

ACM 2643-6817/2026/5-ART
https://doi.org/10.1145/3815787

have also demonstrated the effectiveness of reversible logic in arithmetic operations, highlighting its impact on computational efficiency and circuit optimization [22]. Quantum technology is inherently reversible [65, 94]. Experimental demonstrations of universal quantum gate sets have been reported across several physical platforms, including atomic systems [2], superconducting devices [18, 74], and linear optical systems [19, 92]. The design of multiple-valued logic (MVL) circuits in quantum technology uses the same physical phenomena as binary logic [34].

Multiple-valued logic (MVL) has long attracted attention because of its potential to reduce circuit width, which is a limiting factor in quantum technologies. By encoding more information per quantum digit, MVL enables more compact realizations of quantum circuits, which is crucial for scalability in emerging quantum systems. Compared to binary systems, MVL offers several advantages: improved efficiency in quantum information processing [23], higher memory density, reduced interconnection complexity [10], lower power consumption, and increased fault tolerance [54]. Among MVL systems, ternary logic has been widely studied [14, 15, 17, 73, 86]. However, binary functions cannot always be naturally expressed in ternary form, which imposes limitations. Quaternary logic provides a promising alternative since binary functions can be directly encoded by grouping two bits into one qudit [10, 27, 37, 87]. For example, quaternary logic can be employed to increase the information content of digital signals [62] and improve transmission capacity. A quaternary quantum circuit operates on information units called qudits $\{0, 1, 2, 3\}$, which provide both expressive power and structural compactness [63].

A fundamental motivation for MVL lies in its ability to reduce the number of required quantum information carriers. The general relation between the number of qudits in an m-valued system and qubits in the binary case is:

$$n_m = \frac{1}{\log_2 m} n_2, \tag{1}$$

where n_m is the number of m-valued qudits and n_2 is the number of qubits required to represent the same Hilbert space dimension. This relation highlights the scaling advantage of higher-radix systems: fewer physical carriers of quantum information are required as m increases.

For quaternary systems ($m = 4$), the relation becomes:

$$n_4 = \frac{1}{\log_2 4} n_2 = 0.5\, n_2, \tag{2}$$

indicating that a quaternary system requires exactly half as many qudits as qubits. In other words, quaternary logic achieves a 50% reduction in the number of information carriers compared to binary. Beyond this memory advantage, quaternary logic integrates seamlessly with binary encoding, since two classical bits can be naturally grouped into one qudit. This dual advantage of memory efficiency and binary compatibility makes quaternary systems particularly attractive for scalable quantum architectures. Despite these advantages, quaternary systems face technological challenges, as controlling four quantum levels increases noise sensitivity and coherence loss. In addition, the lack of mature synthesis tools still limits large-scale quaternary circuit design.

Several quaternary reversible circuits have been reported in the literature, including half-adders, full-adders, parallel adders, parallel adders/subtractors, decoders, multiplexers, demultiplexers, subtractors, and comparators [26, 27, 32, 59, 66, 87, 99]. These works highlight the importance of arithmetic building blocks in enabling complex system design. In particular, comparator circuits are indispensable for quantum algorithms, arithmetic units, and decision-making processes [14, 58]. Despite their importance, efficient realizations of quaternary comparators remain limited, as they are often defined manually, and further optimization is required to reduce resource overhead.

The main objective of this paper is to address this gap by presenting a genetic algorithm (GA)-based synthesis framework for quaternary reversible circuits [35] and to demonstrate how it works by focusing on comparator circuits. Our algorithm works by simply providing the truth table of the quaternary circuit and, starting from a

random initial population, it generates a set of candidates at each iteration. The proposed framework evolves candidate circuits based on quaternary Shift and Muthukrishnan–Stroud (M–S) gates, with fitness functions carefully tailored to guide the search toward both correctness and efficiency and by performing mutation and crossover operations. Additionally, our approach implements techniques to prevent stagnation, introduce diversity, and apply elitism to escape from local optima.

We ran our experiments 20 times for each type of quaternary comparators, including 1-qudit equality, less-than, greater-than, and full comparator. Using this approach, we obtain optimized realizations of the different quaternary comparators. We subsequently extend our design to an N-qudit comparator, providing a scalable solution for multi-qudit comparison. The proposed approach emphasizes minimizing the quantum cost, constant inputs, and garbage outputs—three key performance criteria in reversible circuit design. Compared to existing quaternary solutions reported in [32, 66, 99], our method achieves a substantial reduction in resource overhead, leading to more efficient and scalable quaternary quantum architectures: Among all circuits, the proposed designs achieved average reductions of approximately 30% and 58% in quantum cost for the restoring and non-restoring configurations, respectively, compared to the best existing designs reported in the literature. Moreover, we are able to synthesize and optimize the circuits, on average, in only 27 minutes, making our approach applicable in practice.

The remainder of this paper is structured as follows. Section 2 reviews the preliminaries of quaternary reversible logic and gate primitives, as well as on comparator circuits. Section 3 describes the proposed genetic synthesis framework. Section 4 presents the synthesized comparator circuits and N-qudit extension of full comparator. Section 5 evaluates the designs we obtained compared to existing works. Section 6 presents related works, while Section 7 analyzes possible threats to validity. Finally, Section 8 concludes the paper.

2 Preliminaries

In this Section, we introduce the essential concepts and mathematical foundations required for understanding the design and analysis of quaternary quantum logic systems. More specifically, we describe the representation of quaternary quantum states (qudits) in Section 2.1, arithmetic operations in the Galois field GF(4) in Section **??**, and reversible quantum gates specific to quaternary logic such as quaternary Shift and Muthukrishnan–Stroud gates in Section 2.2 and 2.3. These concepts provide the basis for constructing efficient quantum circuits in the quaternary domain. Finally, in Section 2.4, we provide details on quaternary comparator circuits, which are the target of our work.

2.1 Quaternary Logic

Quaternary quantum logic, a subset of multi-valued quantum logic, has gained increasing attention in recent research due to its potential for enhancing computational efficiency and reducing quantum circuit complexity [59, 87]. In this framework, the basic unit of quantum information is referred to as a *qudit* (quantum digit), which, in the case of quaternary systems, can exist in one of four orthonormal basis states: $|0\rangle$, $|1\rangle$, $|2\rangle$, or $|3\rangle$. These basis states are commonly represented by the four 4×1 vectors reported in Equation 3.

$$|0\rangle = \begin{bmatrix}1\\0\\0\\0\end{bmatrix}, \quad |1\rangle = \begin{bmatrix}0\\1\\0\\0\end{bmatrix}, \quad |2\rangle = \begin{bmatrix}0\\0\\1\\0\end{bmatrix}, \quad |3\rangle = \begin{bmatrix}0\\0\\0\\1\end{bmatrix} \tag{3}$$

A qudit in a quaternary quantum system can exist in a linear combination of its basis states, a fundamental property of quantum mechanics known as *superposition*. In quaternary quantum logic, such a superposed state can be expressed as:

$$\psi = \alpha|0\rangle + \beta|1\rangle + \gamma|2\rangle + \delta|3\rangle$$

Here, α, β, γ, and δ are complex probability amplitudes. The square of the magnitude of each coefficient determines the probability of measuring the qudit in the corresponding basis state. Specifically, $|\alpha|^2$, $|\beta|^2$, $|\gamma|^2$, and $|\delta|^2$ represent the probabilities of observing states $|0\rangle$, $|1\rangle$, $|2\rangle$, and $|3\rangle$, respectively.

An N-qudit system can represent 4^N orthonormal basis states, each formed by the tensor product of N individual qudits.

All quantum states in an N-qudit system can be expressed as superpositions over these basis vectors, forming a complete vector space of dimension 4^N [64].

In the context of quaternary quantum logic circuit synthesis, several key metrics are used to evaluate the performance and efficiency of reversible designs [1, 26, 57]:

- *Quantum Cost:* Represents the total cost of a circuit, measured by the number of elementary quaternary 1-qudit Shift and 2-qudit Muthukrishnan–Stroud gates required to implement it [3, 26, 43].
- *Constant Inputs:* Denotes the number of input lines that are initialized with fixed values (0, 1, 2, or 3) during circuit synthesis to facilitate functionality [26, 56].
- *Garbage Outputs:* Refers to the number of output lines that do not contribute to the primary outputs but are necessary to maintain the reversibility of the logic function [26, 51].

Minimization of these parameters is essential for the practical implementation of scalable and resource-efficient quantum circuits in quaternary systems [57]. This ensures lower hardware overhead and reduced error accumulation, making large-scale implementations more feasible.

In this work, we exploit the theory introduced by *quaternary Galois field*, denoted as GF(4). They consists of the set of elements $T = \{0, 1, 2, 3\}$, along with two operations: *addition* ($\oplus$) and *multiplication* ($\odot$) [35]. The behavior of these operations conforms to the algebraic structure defined in Tables 1 and 2, and satisfies the standard field properties.

Table 1. Addition table in GF(4).

$\oplus$	**0**	**1**	**2**	**3**
0	0	1	2	3
1	1	0	3	2
2	2	3	0	1
3	3	2	1	0

2.2 Quaternary Shift Gates

In quaternary reversible logic, every transformation applied to a qudit is represented by a 4×4 unitary matrix [31], as illustrated in Figure 1. These matrices consist of 0, 1, 2 and 3 in each row and column containing exactly four elements. The transformations encoded by these matrices define how qudit states are shifted or swapped. Specifically, the matrix Z(+0) represents the identity transformation, where the qudits remains unchanged. Each row or column of this matrix corresponds to one of the quaternary basis states: $|0\rangle$, $|1\rangle$, $|2\rangle$, or $|3\rangle$, as detailed in Equation 3. The transformation Z(+1) and Z(+2) perform a forward cyclic shift on the qudit states $|1\rangle$ and $|2\rangle$,

Table 2. Multiplication table in GF(4).

⊙	**0**	**1**	**2**	**3**
0	0	0	0	0
1	0	1	2	3
2	0	2	3	1
3	0	3	1	2

respectively, while Z(+3) applies a similar shift to the state $|3\rangle$. Other transformations denote specific pairwise exchanges of qudit states, for instance, Z(01) swaps $|0\rangle$ and $|1\rangle$. These matrices correspond to 1-qudit Shift gates, which are fundamental operations in quaternary logic. The symbol representation of these quaternary Shift gates is provided in Figure 2. Each gate performs a transformation by applying a Z operation to the input qudit A, producing an output P. These gates are compatible with implementation on liquid ion trap quantum platforms [9, 75] and have a quantum cost of 1. We note that different implementations of quantum gates, e.g., optical technologies [16, 49], nonadiabatic geometric phases [100] or semi-conductor based [84], may have different costs. In this work, however, we use the commonly used high-level abstraction in which shift gates are treated as unit-cost gates. The inverse gates corresponding to the 1-qudit Shift gates are summarized in Table 3.

Table 3. Inverse mapping of the quaternary 1-qudit Shift gates.

Gate	Z(+0)	Z(+1)	Z(+2)	Z(+3)	Z(123)	Z(013)
Inverse	Z(+0)	Z(+1)	Z(+2)	Z(+3)	Z(132)	Z(031)
Gate	Z(021)	Z(032)	Z(132)	Z(012)	Z(023)	Z(031)
Inverse	Z(012)	Z(023)	Z(123)	Z(021)	Z(032)	Z(013)
Gate	Z(23)	Z(01)	Z(0213)	Z(0312)	Z(12)	Z(0132)
Inverse	Z(23)	Z(01)	Z(0312)	Z(0213)	Z(12)	Z(0231)
Gate	Z(0231)	Z(03)	Z(13)	Z(0123)	Z(02)	Z(0321)
Inverse	Z(0132)	Z(03)	Z(13)	Z(0321)	Z(02)	Z(0123)

2.3 Quaternary Muthukrishnan and Stroud Gates

Muthukrishnan and Stroud introduced a family of 2-qudit gates, which are realizable using quantum technologies such as liquid ion traps [61]. These gates form a foundational component for constructing multivalued quantum circuits. The Muthukrishnan–Stroud gate, illustrated in Figure 3, features conditional behavior controlled by the input qudits: A (the control) and B (the target). The gate outputs P and Q, where P replicates the control input A, and Q reflects a transformation of B determined by a Z-transformation. This transformation, selected from a defined set shown in Figure 1, is applied only when A is in the logical state $|3\rangle$. If A is in any other state, the gate leaves B unchanged in the target output. This gate has a quantum cost of 1. This follows the common abstraction in the literature, where it is treated as a unit-cost primitive for the purpose of circuit-level evaluation and fair comparison. From an implementation perspective, these gates are theoretically realizable in physical platforms supporting multi-level quantum states (e.g., trapped-ion systems). However, their practical realization and decomposition into elementary operations remain challenging and depend on the underlying technology. As a consequence, the practical cost of such gates may be different from the unitary one we consider in this work.

$$Z(0)=\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&1\end{bmatrix} \quad Z(+1)=\begin{bmatrix}0&1&0&0\\1&0&0&0\\0&0&0&1\\0&0&1&0\end{bmatrix} \quad Z(+2)=\begin{bmatrix}0&0&1&0\\0&0&0&1\\1&0&0&0\\0&1&0&0\end{bmatrix} \quad Z(+3)=\begin{bmatrix}0&0&0&1\\0&0&1&0\\0&1&0&0\\1&0&0&0\end{bmatrix}$$

$$Z(123)=\begin{bmatrix}1&0&0&0\\0&0&0&1\\0&1&0&0\\0&0&1&0\end{bmatrix} \quad Z(013)=\begin{bmatrix}0&0&0&1\\1&0&0&0\\0&0&1&0\\0&1&0&0\end{bmatrix} \quad Z(021)=\begin{bmatrix}0&1&0&0\\0&0&1&0\\1&0&0&0\\0&0&0&1\end{bmatrix} \quad Z(032)=\begin{bmatrix}0&0&1&0\\0&1&0&0\\0&0&0&1\\1&0&0&0\end{bmatrix}$$

$$Z(132)=\begin{bmatrix}1&0&0&0\\0&0&1&0\\0&0&0&1\\0&1&0&0\end{bmatrix} \quad Z(012)=\begin{bmatrix}0&0&1&0\\1&0&0&0\\0&1&0&0\\0&0&0&1\end{bmatrix} \quad Z(023)=\begin{bmatrix}0&0&0&1\\0&1&0&0\\1&0&0&0\\0&0&1&0\end{bmatrix} \quad Z(031)=\begin{bmatrix}0&1&0&0\\0&0&0&1\\0&0&1&0\\1&0&0&0\end{bmatrix}$$

$$Z(23)=\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&0&1\\0&0&1&0\end{bmatrix} \quad Z(01)=\begin{bmatrix}0&1&0&0\\1&0&0&0\\0&0&1&0\\0&0&0&1\end{bmatrix} \quad Z(0213)=\begin{bmatrix}0&0&0&1\\0&0&1&0\\1&0&0&0\\0&1&0&0\end{bmatrix} \quad Z(0312)=\begin{bmatrix}0&0&1&0\\0&0&0&1\\0&1&0&0\\1&0&0&0\end{bmatrix}$$

$$Z(12)=\begin{bmatrix}1&0&0&0\\0&0&1&0\\0&1&0&0\\0&0&0&1\end{bmatrix} \quad Z(0132)=\begin{bmatrix}0&0&1&0\\1&0&0&0\\0&0&0&1\\0&1&0&0\end{bmatrix} \quad Z(0231)=\begin{bmatrix}0&1&0&0\\0&0&0&1\\1&0&0&0\\0&0&1&0\end{bmatrix} \quad Z(03)=\begin{bmatrix}0&0&0&1\\0&1&0&0\\0&0&1&0\\1&0&0&0\end{bmatrix}$$

$$Z(13)=\begin{bmatrix}1&0&0&0\\0&0&0&1\\0&0&1&0\\0&1&0&0\end{bmatrix} \quad Z(0123)=\begin{bmatrix}0&0&0&1\\1&0&0&0\\0&1&0&0\\0&0&1&0\end{bmatrix} \quad Z(02)=\begin{bmatrix}0&0&1&0\\0&1&0&0\\1&0&0&0\\0&0&0&1\end{bmatrix} \quad Z(0321)=\begin{bmatrix}0&1&0&0\\0&0&1&0\\0&0&0&1\\1&0&0&0\end{bmatrix}$$

Fig. 1. The representation of 1-qudit permutative transforms.

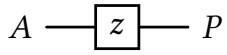


Fig. 2. The symbolic representation of quaternary Shift gates.

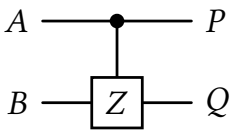


Fig. 3. The symbolic representation of quaternary Muthukrishnan and Stroud gate.

2.4 Quaternary Comparator Circuits

In this section, we describe the comparators we use as case studies to demonstrate our GA-based synthesis and optimization approach. More specifically, we use the less-than comparator (Section 2.4.1), the greater-than comparator (Section 2.4.2), the equality comparator (Section 2.4.3), and the full comparator (Section 2.4.4).

Each comparator circuit operates on three quantum lines: two qudit inputs, A and B, and one ancillary line initialized to a known constant. In the output, the ancilla line is overwritten with the comparison result, which is the target output. This design ensures the circuit remains reversible, with a one-to-one mapping between input and output states. As A and B each take values in $\{0, 1, 2, 3\}$, there are 16 unique input states for each comparator. In the following, we better describe the function of each of the comparators we analyze.

2.4.1 Less-than Comparator. This circuit is designed to compare the two inputs A and B and check whether $A < B$. More specifically, the output L is as follows:

$$L(A,B) = \begin{cases} 1 & \text{if } A < B \\ 0 & \text{otherwise} \end{cases}$$

The realization of the less-than comparator requires identifying exactly six input states in which the inequality holds. The final circuit typically includes several Shift, and Muthukrishnan and Stroud gates, triggered by patterns where $A < B$.

2.4.2 Greater-than Comparator. This circuit is designed to compare the two inputs A and B and check whether $A > B$. More specifically, the output G is as follows:

$$G(A,B) = \begin{cases} 2 & \text{if } A > B \\ 0 & \text{otherwise} \end{cases}$$

As with the less-than circuit, only six input states trigger the transformation of the ancilla line. This circuit contains gate sequences that distinguish these inputs through conditional logic encoded by Shift, and Muthukrishnan and Stroud gates.

2.4.3 Equality Comparator. This circuit is designed to compare the two inputs A and B and check whether they are equivalent. More specifically, the output E is as follows:

$$E(A,B) = \begin{cases} 3 & \text{if } A = B \\ 0 & \text{otherwise} \end{cases}$$

2.4.4 Full Comparator. This circuit is designed to compare the two inputs A and B by providing all functionalities described by the previous circuits. More specifically, the output F is as follows:

$$F(A,B) = \begin{cases} 1 & \text{if } A < B \\ 2 & \text{if } A > B \\ 3 & \text{if } A = B \end{cases}$$

This mapping covers the entire 16 state space uniquely, making it ideal for unified comparison logic in quantum algorithms such as Grover's search [97]. The circuit for this comparator is more complex, often utilizing both Shift, and Muthukrishnan and Stroud gates to direct the ancilla line to the correct final state.

3 Proposed Genetic Synthesis Framework

Synthesizing optimized reversible circuits from truth tables is particularly challenging in quaternary logic, where balancing correctness and compactness is difficult. For this reason, in this paper, we explore the use of genetic algorithms (GA) to automatically synthesize a circuit given its truth table. A GA-based framework offers an automated search strategy to generate efficient and functionally accurate designs.

This section presents the proposed GA-based framework for synthesizing compact and functionally accurate reversible comparators in quaternary logic. Unlike binary or ternary logic, quaternary systems operate on quantum states with four distinct levels, known as qudit, which offer enhanced data density and richer computational expressiveness. The methodology we propose in this paper is capable of synthesizing quaternary circuits that implement comparison operations for less than, greater than, equal to, as well as a 1-qudit full comparator that detects all three relations. While in this work we focus on the synthesis of reversible quaternary comparators as case studies, the proposed methodology is general: In principle, any quaternary reversible circuit can be synthesized, provided that its truth table is available.

The genetic synthesis approach is based on evolutionary computation principles, where a population of reversible circuits evolves over time using biologically inspired operators such as mutation, crossover, and selection. The ultimate goal is to discover optimized quaternary circuits that match predefined truth tables, minimize circuit complexity, and preserve reversibility. Indeed, the process we present in this paper preserves reversibility by confining the synthesis process to compositions of reversible quaternary gates. The scrips implementing the proposed GA-based synthesis are available in our replication package at https://github.com/foselab/QuantumCircuitsGeneticSynthesisAndOptimization.

Our approach is implemented by exploiting the functionalities offered by the jMetalPy framework [5]. The process we implement in our GA-based framework is reported in Figure 4 and described in details for each step in the following subsections.

3.1 Circuit Representation and Chromosome Encoding

Each individual (chromosome) in the GA represents a potential reversible circuit encoded as an ordered sequence of reversible gates. A gene in the chromosome is defined as a triplet:

$$(control\_line, target\_line, gate\_type)$$

The gates are drawn from a rich set of quaternary Shift Gates and their controlled counterparts, quaternary Muthukrishnan and Stroud Gates, where the action of the gate is triggered only if the control line is set to logic level 3. A gate is interpreted as uncontrolled when its control and target lines are identical. The gate library used consists of 24 unique permutations derived from GF(4), corresponding to all possible bijective mappings of the four logic levels (i.e., all permutations of the set {0,1,2,3}), as illustrated in Figure 1. These transformations ensure reversibility by preserving a one-to-one mapping between inputs and outputs. These permutations include cyclic permutations, reflections, and linear transformations over the quaternary domain.

3.2 Genetic Algorithm Initialization

To design functionally accurate quaternary reversible comparator circuits, a precise specification of the target behavior is essential (Step 1 in Figure 4). During this phase, we define the truth table of the circuit to be synthesized. We emphasize that the algorithm we implement is general and can work with any quaternary circuit, provided its truth table. However, to demonstrate the usefulness and describe the process we implemented, we rely on the circuits previously introduced in Section 2.4. An example of the Python code implementing the full comparator (see Section 2.4.4) is reported in Listing 1.

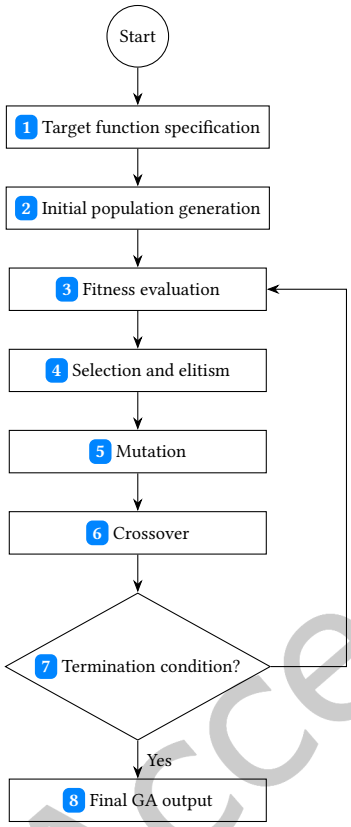


Fig. 4. Flowchart of the GA-based synthesis process.

Listing 1. Python code implementing the truth table of the 1-qudit full comparator.

```
def generate_truth_table():
    table = {}
    for a in range(4):
        for b in range(4):
            if a == b:
                f = 3
            elif a > b:
                f = 2
            else:
                f = 1
            table[(a, b, 0)] = (a, b, f)
    return table
```

After having defined the target functions, Step 2 in Figure 4 requires one to define the initial population from which to start the evolution. In our case, we define a population of 1,000 random individuals, each of them with 1

to 50 random gates chosen from those corresponding to the matrices presented in Figure 1 or their controlled version.

3.3 Handling Don't-Care (DC) Conditions

Many practical reversible or quantum circuits correspond to incompletely specified functions, where certain input combinations are either physically unreachable or functionally irrelevant. To efficiently synthesize such circuits, our framework supports the definition of a don't-care (DC) set within the truth-table specification.

From an algorithmic perspective, handling DC conditions requires balancing two competing aspects: (1) the increased search flexibility that allows the genetic algorithm (GA) to explore more compact and low-cost circuit realizations, and (2) the computational cost of evaluating individuals over an expanded search space.

In the proposed framework, the DC set is explicitly defined together with the specified input–output mappings. During the fitness-evaluation phase, the outputs corresponding to the DC input combinations are not required to match a specific target value; instead, any valid reversible mapping is accepted. This approach allows the GA to assign arbitrary output values to DC cases, effectively enlarging the solution space and improving the probability of discovering simpler circuits.

However, to prevent excessive exploration time, DC conditions are considered selectively. Specifically, when the fraction of DC entries exceeds a predefined threshold, a partial-evaluation strategy is used: only a random subset of DC combinations is sampled at each generation. This stochastic handling of DCs preserves diversity and reduces runtime while maintaining the optimization benefits of incompletely specified functions. This mechanism provides a controlled way to exploit DC conditions, ensuring that synthesis remains both efficient and scalable, while still leveraging the flexibility of partially specified functions to generate compact reversible circuits.

The exploitation of DCs in reversible and quantum circuit synthesis has been investigated in prior work, particularly in combination with heuristic and evolutionary techniques. For instance, the authors of [56] demonstrate that DC assignments can significantly reduce quantum cost by enlarging the feasible design space, while approaches such as the one proposed by [69] incorporate DCs by iteratively completing unspecified entries and synthesizing fully specified instances. In these methods, DC are typically handled outside the core synthesis loop, either through pre-processing or by exploring alternative completions of the specification. Differently from these approaches, our method embeds the DC set directly into the evolutionary loop by treating unspecified input combinations as unconstrained during fitness evaluation, thus allowing the genetic algorithm to implicitly determine favorable assignments while exploring the search space. This avoids repeated re-synthesis of fully specified instances and enables a tighter integration between DC exploitation and circuit optimization.

3.4 Fitness Function Evaluation

At Step 3 of the process depicted in Figure 4, each individual is evaluated by means of a fitness function. The purpose of this function is to drive the GA to generate circuits that are closer to behave as required by the chosen truth table and, when multiple correct circuits are available, to prefer those minimizing the quantum cost. Note that we consider the quantum cost solely in terms of the number of gates. Although the numbers of constant inputs and garbage outputs could, in principle, also affect the cost of a quaternary circuit, they are fixed by the user at the start of the algorithm and remain unchanged throughout the genetic synthesis process.

To implement this behavior, we designed our fitness function as a piece-wise function. In particular, let $F_T(A, B)$ be the target output function, $F'_i(A, B)$ be the output function of the individual i, and $cost(i)$ the quantum cost for i, the fitness $fit(i)$ is defined as follows:

$$fit(i) = \begin{cases} \dfrac{\|\forall_{A\in T,B\in T}\ F_T(A,B) = F'_i(A,B)\|}{\|A\| \cdot \|B\|} & \text{if } \exists_{A,B\in T}\ F_T(A,B) \neq F'_i(A,B) \\ 1 + \dfrac{1}{cost(i)} & \text{if } \forall_{A,B\in T}\ F_T(A,B) = F'_i(A,B) \end{cases}$$

Intuitively, this function distinguishes the scenario in which a correct circuit has not been found yet ($\exists_{A,B\in T}\ F_T(A,B) \neq F'_i(A,B)$, i.e., there is at least one input combination for which the expected output does not match the obtained one) and the one in which the circuit is correct ($\forall_{A,B\in T}\ F_T(A,B) = F'_i(A,B)$), but it can be optimized. In the former case, $fit(i)$ computes the ratio between the number of correct outputs ($\|\forall_{A\in T,B\in T}\ F_T(A,B) = F'_i(A,B)\|$) and the total number of outputs ($\|A\| \cdot \|B\|$). In the latter, it adds to this factor the $cost(i)$ of the individual i under consideration. $fit(i) < 1$ indicates an incorrect circuit, while $fit(i) \geq 1$ is obtained for a correct circuit: The higher the fitness, the better the circuit. Indeed, in our GA, we aim to maximize this fitness, i.e., we try to drive the circuit under evolution to make it behave as in the target truth table and, when it happens, to reduce its cost. This approach is commonly adopted in the literature on multiple-valued circuit synthesis. For example, in [46] and [82] the fitness function is a combination of functional correctness and structural cost (e.g., gate count or quantum cost), where individuals are penalized until exact functional matching is achieved, and then further optimized through cost-reduction terms. Some work include additional factors within the fitness function beyond the pure number of gates, such as the number of garbage outputs and ancilla lines [14], as well as circuit depth [95].

3.5 Genetic Operators and Features

This section details the genetic operators we implement in our GA-based quantum circuit synthesis approach. More specifically, we present the selection and elitism (Section 3.5.1), mutation (Section 3.5.2), and crossover (Section 3.5.3) mechanisms. Finally, we discuss the methods we use to prevent stagnation (Section 3.5.4).

3.5.1 Selection and Elitism. Identifying the appropriate selection technique is a critical step in a GA. The process of selection (Step 4 in Figure 4) plays an important role in resolving premature convergence because it occurs due to lack of diversity in the population. Therefore, selection of population in each generation is very important [83].

For our GA-based approach for the synthesis and optimization of quantum circuits, we defined the ElitistGeneticAlgorithm class, which extends the GeneticAlgorithm class provided by jMetalPy by implementing elitism [76], diversity injection [53], and stagnation avoidance.

The pseudo-code of the function selecting individuals for the next evolution iteration is reported in Algorithm 1. It works on a *population* of individuals which are evolved in new circuits (see the following sections) in the *offspring* set. During the first iteration, the *offspring* set is empty, while in the following iterations it contains the circuits obtained through mutation (Step 5 in Figure 4) and crossover (Step 6 in Figure 4).

At the beginning of the selection process, a new population *new_pop* is built (line 1) by merging the previous population and the offspring. In order to prefer more fitter solutions, the new population is sorted by fitness (line 2) and the best fitness value is extracted (line 3). Then, the algorithm checks whether the process is stagnating, by considering if the new best fitness is improving the old one by a factor greater than the set stagnation threshold ε (lines 4-9).

If the evolution process stagnates more than the stagnation limit s (line 10), the top e_r individuals (i.e., the elite) are extracted from the population (line 11) and a set of *diverse* solutions is generated (lines 12-16). The size of this set is determined by the highest fitness achieved in the population *new_pop*. If the highest fitness surpasses τ, the diverse set comprises 80% of the number of circuits in the population; otherwise, it contains 30% of the number of circuits. By introducing this approach, we enhance diversity in scenarios where fitter circuits are experiencing stagnation and are unable to be transitioned into more optimal or correct ones. Finally, the

Algorithm 1 Selection operation.

Require: $population$, the population of quantum circuits
Require: $offspring$, the offspring
Require: e_r, the elite rate
Require: ε, the threshold for stagnation
Require: s, the stagnation limit
Require: τ, the fitness threshold for diversity injection
Require: p_s, the population size

1: $new\_pop \leftarrow population + offspring$
2: new_pop.sortByFitness()
3: $bestFit \leftarrow new\_pop$.getHighestFitness()
4: **if** $bestFit \leq previous\_best + \varepsilon$ **then** ▷ Check if the fitness is not improving
5: $stagnation\_count \leftarrow stagnation\_count + 1$
6: **else**
7: $previous\_best \leftarrow bestFit$
8: $stagnation\_count \leftarrow 0$
9: **end if**
10: **if** $stagnation\_count > s$ **then** ▷ The evolution process is stagnating
11: $elite \leftarrow new\_pop[0, e_r \cdot new\_pop$.size()$]$
12: **if** $bestFit > \tau$ **then**
13: $diverse \leftarrow$ Generate diverse solutions covering 80% of the population
14: **else**
15: $diverse \leftarrow$ Generate diverse solutions covering 30% of the population
16: **end if**
17: $new\_pop \leftarrow elite + diverse + new\_pop[e_r \cdot new\_pop$.size(), $p_s - e_r \cdot new\_pop$.size()$]$
18: **else**
19: $new\_pop \leftarrow new\_pop[0, p_s]$
20: **end if**
21: **return** new_pop

new population is built (line 17) by merging the *elite* and the *diverse* sets with the previous population, with a limit of p_s individuals. Alternatively, if no stagnation is revealed, a the top p_s individuals are extracted from the population (line 19 and returned as the new population new_pop (line 21).

3.5.2 Mutation. In a GA, mutation (Step 5 in Figure 4) operators apply some changes randomly to one or more genes to produce new offspring, in order to create new solutions with enhanced fitness. Mutations play a crucial role in preventing the GA from getting stuck in a local optimum.

For our GA-based synthesis and optimization of quantum circuits, we defined the CircuitMutation class, which extends the Mutation class provided by jMetalPy. It implements 6 different mutation types:

- Add: It adds in a random position a new gate, randomly chosen between those supported by the specific multi-valued logic;
- Remove: It removes a random gate from the circuit;
- Change: it changes a gate with a new one, randomly chosen between those supported by the specific multi-valued logic;
- Swap: It randomly selects two gates and swaps them in the circuit;
- Split: It removes the initial and the final part of a circuit, in order to obtain a sub-circuit. The lengths of the two parts to be removed are randomly chosen;
- Optimize: It performs operations that do not change the circuit behavior but make it more compact. This mutation is inspired by local circuit rewriting and simplification strategies commonly used in quantum and

reversible logic synthesis. The primary objective is to eliminate redundant operations, reduce gate count, and enhance the overall implementation efficiency in physical quantum systems. Thus, the optimizations that are performed when this mutation type is chosen are as follows:

- Duplicate and Inverse Gate Elimination: Pairs of gates that are inverses of each other (a gate followed immediately by its inverse) are removed, as they have no effect on the quantum state. This rule also applies to multi-gate patterns that evaluate to identity transformations.
- Pruning of Inactive or Irrelevant Gates: If a gate does not influence the target output, performs no meaningful operation (such as an identity transformation), or requires a control value that cannot be reached within the circuit, it is removed;
- Merging Controlled Gate Sequences: Sequences of Muthukrishnan-Stroud (M-S) gates acting on the same control line are analyzed for combinability. If multiple M-S gates with identical control lines can be merged into a single equivalent gate, the transformation is applied to simplify the structure.
- Collapsing Redundant Shift Gates: If two or more 1-qudit Shift gates (gates acting on a single line without control) are found consecutively on the same target line, and their combined action is equivalent to a single permutation, they are replaced by the corresponding composite 1-qudit Shift gate.

The pseudo-code of the function performing the mutation operation is reported in Algorithm 2. Our CircuitMutation starts from a circuit (*circuit*) and generates a new candidate. To introduce randomness in the mutation mechanism, we require the user to specify the mutation rate m_r for its application (line 1). To improve the efficacy of our approach, we use an adaptive mutation rate throughout the evolution process. More specifically, we adapt the mutation rate based on the fitness thanks to the following formula, where $\tilde{m}_r$ is the mutation rate chosen by the user, and $fit(i)$ is the fitness of the individual under mutation:

$$m_r = \tilde{m}_r \cdot \left(1 - \frac{fit(i)}{2}\right)$$

The purpose of the adaptive rate is to mutate more likely individuals with lower fitness.

Then, if mutation can be applied, we define the possible mutations to be applied to *circuit*, based on the current individual status (from line 5 to line 15). More specifically, if new gates can be added (line 5) the Add mutation is considered; If the number of gates is higher than the minimum allowed (line 8) the Remove mutation is considered; If the circuit contains at least one gate (line 11) the Swap, Change, and Split mutations are considered; Finally, if the circuit is behaviorally correct (line 14) we also consider the Optimize mutation. When the set of possible mutations has been built, our algorithm extracts a random mutation (line 17) and applies the chosen mutation to the circuit (line 18).

3.5.3 Crossover. The crossover (Step 6 in Figure 4) is an imitation of reproduction in biological beings. Crossover exchanges information between different individuals to generate offspring with the hope of obtaining better genes [67].

Our approach defines the CircuitCrossover class which extends the Crossover class provided by jMetalPy by adding the support to quantum circuits. The pseudo-code of the function performing the crossover operation is reported in Algorithm 3.

Our CircuitCrossover starts from two circuits ($parent1$ and $parent2$) and generates two offspring. To introduce randomness in the crossover mechanism, we require the user to specify a the crossover rate c_r for its application (line 1). If crossover can be applied and each of the parents has at least one gate (line 4), we extract two random indexes, namely i_1 and i_2, included between 1 and the length of each circuit (line 5 and line 6). Then, the crossover is applied: Two offspring $child1$ and $child2$ are generated. The former will be composed by the first part of $parent1$ and the second part of $parent2$, while the latter will be composed by the first portion of $parent2$ and the

Algorithm 2 Mutation operation.

Require: $circuit$, the circuit to be mutated
Require: m_r, the mutation rate
Require: $MaxGenes$, the maximum number of genes, i.e., of gates
Require: $MinGenes$, the minimum number of genes, i.e., of gates
Ensure: $mutatedCircuit$, the mutated circuit

if random() > m_r **then**
  return $circuit$
end if
$mutations \leftarrow \emptyset$
if length($circuit$) < $MaxGenes$ **then** ▷ If possible, consider the Add mutation
  $mutations$.add('Add')
end if
if length($circuit$) > $MinGenes$ **then** ▷ If possible, consider the Remove mutation
  $mutations$.add('Remove')
end if
if length($circuit$) > 0 **then** ▷ If possible, consider the Swap, Change, and Split mutations
  $mutations$.add(['Swap', 'Change', 'Split'])
end if
if $circuit$.isCorrect() **then** ▷ If the circuit is correct, consider the Optimize mutation
  $mutations$.add('Optimize')
end if
$i \leftarrow$ random(1, length($mutations$))
$mutatedCircuit \leftarrow circuit$.applyMutation($mutations(i)$) ▷ Apply the chosen mutation
return $mutatedCircuit$

Algorithm 3 Crossover operation.

Require: $parent1$, the first parent
Require: $parent2$, the second parent
Require: c_r, the crossover rate
Require: $MaxGenes$, the maximum number of genes, i.e., of gates

if random() > c_r **then**
  return $parent1$, $parent2$
end if
if length($parent1$) > 1 & length($parent2$) > 1 **then**
  $i_1 \leftarrow$ random(1, length($parent1$))
  $i_2 \leftarrow$ random(1, length($parent2$))
  $child1 \leftarrow parent1[0{:}i_1] + parent2[i_2{:}\dots]$
  $child2 \leftarrow parent2[0{:}i_2] + parent1[i_1{:}\dots]$
  if length($child1$) > $MaxGenes$ **then**
    $child1 \leftarrow child1[0{:}MaxGenes]$
  end if
  if length($child2$) > $MaxGenes$ **then**
    $child2 \leftarrow child2[0{:}MaxGenes]$
  end if
  return $child1$, $child2$
end if
return $parent1$, $parent2$

second of *parent*1 (line 7 and line 8). If the obtained circuit is longer than the maximum number of genes set by the user (*MaxGenes*), the part in excess is cut (line 9 to line 14).

3.5.4 Stagnation prevention. A key challenge in evolutionary synthesis is avoiding stagnation, which occurs when the search process converges prematurely to suboptimal solutions, generally corresponding to local optimum. To address this, the proposed genetic framework integrates multiple complementary mechanisms that actively promote exploration. First, stagnation is explicitly detected by monitoring improvements in fitness across generations; when progress falls below a threshold for a predefined number of iterations, corrective actions are triggered. In such cases, *diversity injection* is performed by introducing a significant portion of newly generated random individuals into the population, thereby expanding the search space and enabling the discovery of alternative solutions. More specifically, while normally only 20% of the individuals are replaced among iterations, when the process stagnates for too many iterations and the fitness improvement is too low, 80% of the individuals are replaced by random ones. This approach is normally referred to in the literature as *random restarts* [21]. At the same time, elitism ensures that the best-performing circuits are preserved, maintaining convergence pressure toward high-quality designs. Mutation operators further contribute by introducing structural variations [13]. Together, these strategies create a dynamic balance between exploration and exploitation, allowing the algorithm to escape local optima and continue progressing toward more compact and efficient circuit implementations. Nevertheless, further improvements are possible, and future work will consider additional techniques, such as Simulated Annealing [7] or adaptive mutation rates [90], to further enhance global search capabilities. Similarly, we plan to introduce directed mutations [8] to purposefully steer individuals towards better fitness areas and enhance search efficiency.

3.6 Evolution Process and Termination Criteria

Starting from an initial population, our GA-based approach evolves a set of candidates by repeating steps 4, 5, and 6 in Figure 4 until the termination condition (7 in Figure 4) is met.

We emphasize that the GA and the fitness we designed are, theoretically, unbounded: the fitness could increase up to ∞, since it depends on the inverse of the number of gates. For this reason, setting a target fitness and terminate the evolution process when one of the individuals reaches that value is not possible. Thus, we exploit the StoppingByEvaluations class offered by jMetalPy which allows users to set the maximum number of evaluations to be performed during each run. We compute that number by multiplying the population size for the number of generations.

When the number of evaluations is reached, the best candidate obtained throughout the entire GA process is returned as the result (8 in Figure 4).

4 Synthesized Circuits Based on the Proposed Genetic Algorithm

This section evaluates our GA-based approach for quantum circuit synthesis and optimization by presenting the circuits we obtained. We configured the GA parameters with the values reported in Table 4. For each of the circuits under analysis (see Section 2.4), we executed 20 runs of 50,000 generations each. This number of runs follows standard practice in evolutionary computation studies, where 15–30 independent repetitions are typically employed to obtain statistically meaningful results [24, 44, 91]. In each generation, a population of 100 individuals is used, among of which 10 are considered as the elite. Each individual contains between 1 and 50 genes, i.e., quantum gates. During the evolution process, mutation is applied to an individual with a probability of 20% while crossover with a probability of 70%. To prevent stagnation (see Section 3.5.1), after 15 generations with non increasing fitness, we inject random circuits to improve the diversity of the solutions. We consider stagnation an improvement in fitness of less than 10^{-6} between two consecutive generations. Finally, to decide the magnitude of diversity injection, we consider a threshold in fitness of 0.8.

Parameter	Value
Number of runs	20
Population size (p_s)	1,000
Elite size (e_r)	10
Generations	50,000
Mutation rate ($\tilde{m}_r$)	0.2
Crossover rate (c_r)	0.7
Stagnation limit (s)	15
Stagnation threshold (ε)	10^{-6}
Fitness threshold for diversity (τ)	0.8
Minimum number of gates per circuit (*MaxGenes*)	1
Maximum number of gates per circuit (*MinGenes*)	50

Table 4. Configuration of the GA.

On average, each run of our experiments required 27 minutes on a PC running Ubuntu server 24.04.2, with 256GB RAM, AMD Ryzen Threadripper PRO 7985WX 64-Cores CPU, and a GeForce RTX 4090 GPU.

In the following, we present the design of 1-qudit less-than, greater-than, equality, and full comparators, which have been derived thanks to the proposed GA-based synthesis and optimization approach. In addition, we show how the 1-qutit full comparator can be generalized to a N-qudit full comparator, by just adding a simple sub-comparator that we designed thanks to the proposed GA-based approach.

It should be noted that all circuits synthesized by the proposed genetic algorithm correspond to the *non-restoring* form. Although the designs are reversible, the input qudits A and B are not preserved, as they are transformed into the garbage outputs P and Q, while the comparator flag F is produced as the functional output. In many applications this behavior is acceptable, since only F is required. Nevertheless, a *restoring* variant can be systematically derived by applying the inverse of the transformation (see Table 3) on the first two lines once F has been computed. This follows the standard compute–uncompute paradigm [68] in reversible circuit design, and does not alter the role of the genetic algorithm, which provides the initial optimized structure. In this way, both a compact non-restoring version and a restoring version of each comparator can be realized, depending on the target application. Furthermore, all circuits synthesized by the proposed GA-based framework assume that constant qudits are initialized to the logical state 0. This choice aligns with standard reversible synthesis assumptions and reflects the default initialization in current hardware. Using other initial states would require additional state-preparation gates, thereby increasing the overall quantum cost.

4.1 Less-than Comparator

In this section, we present the less-than comparator circuit (see Section 2.4.1) synthesized and optimized by our GA-based framework. Across 20 independent GA runs, the optimization process demonstrated complete consistency, with all runs converging to the same optimal solution characterized by a best fitness value of 1.142857 and a circuit length of 7 gates. This uniform convergence indicates that the GA reliably discovered the minimal and fully correct less-than comparator circuit across all runs, without any suboptimal or divergent outcomes. Figure 5 illustrates the identical fitness values obtained in each run, confirming the stability and determinism of the proposed evolutionary synthesis process for this configuration.

The best solution identified for the less-than comparator had a length of 7 and a fitness of 1.142857. This compact design is reported in Figure 6, where the inputs consist of qudits A, B, and a constant initialized to 0. The

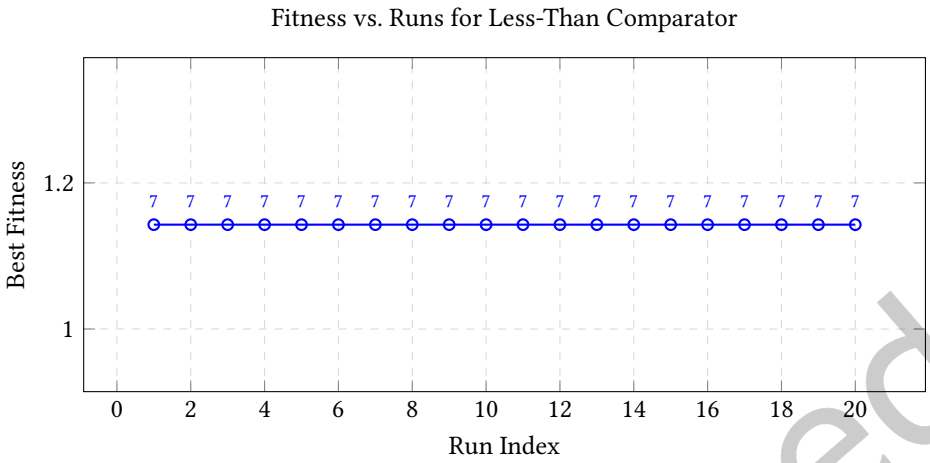


Fig. 5. Fitness across 20 GA runs for the 1-qudit less-than comparator. Each point is the best circuit of a run; labels indicate circuit length.

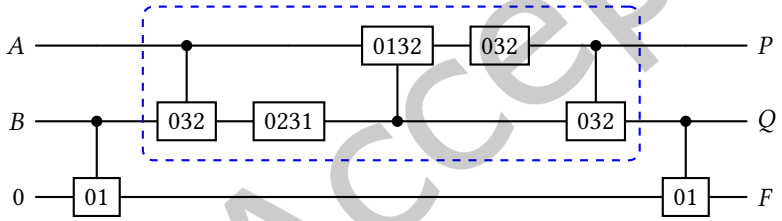


Fig. 6. Gate-level implementation of 1-qudit less-than comparator circuit using the proposed genetic algorithm.

outputs are P, Q, and F, with F representing the comparator result that indicates the outcome of the comparison between the inputs.

As observed, the proposed less-than comparator consists of one constant input with the value of zero and two garbage outputs, which we denote with P and Q. The quantum cost of this circuit is 7, representing a substantial improvement over the designs previously reported in [66], [99], and [32], where the quantum costs are 22, 25, and 39, respectively. Furthermore, our design utilizes only 2 garbage outputs and 1 constant input, in contrast to all existing designs which require 3 garbage outputs and 2 constant inputs [32, 66, 99]. If input restoration is required, the circuit can be extended according to the general procedure described at the beginning of this section, by applying the inverse of the gates highlighted in the blue box in this figure, resulting in a restoring inputs with a quantum cost of 12, which is still substantially lower than the costs reported in [32, 66, 99].

4.2 Greater-than Comparator

In this section, we present the greater-than comparator circuit (see Section 2.4.2) synthesized and optimized by our GA-based framework. Across 20 independent runs, the best fitness values were highly consistent, ranging between 1.12 and 1.14, corresponding to correct circuit lengths of 7 or 8 gates. The majority of runs (17 out of 20) converged to the optimal 7-gate solution with a fitness of 1.142857, while three runs yielded slightly longer 8-gate

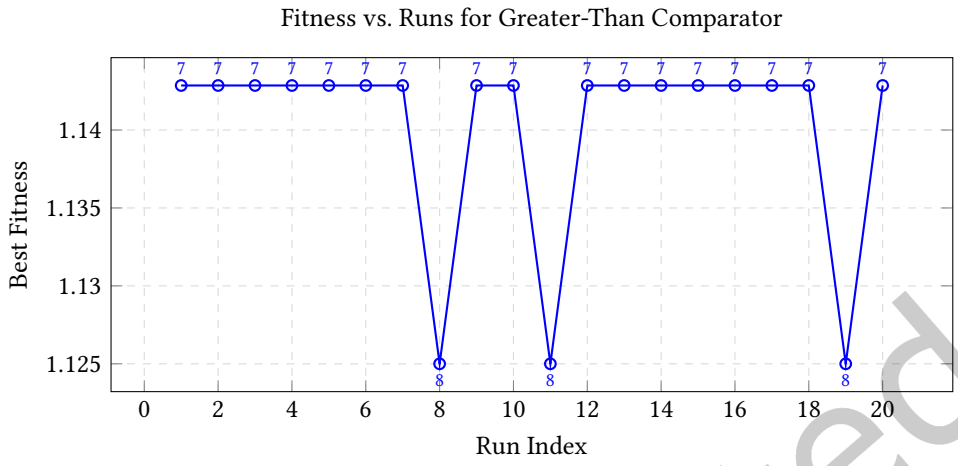


Fig. 7. Fitness across 20 GA runs for the 1-qudit greater-than comparator. Each point is the best circuit of a run; labels indicate circuit length.

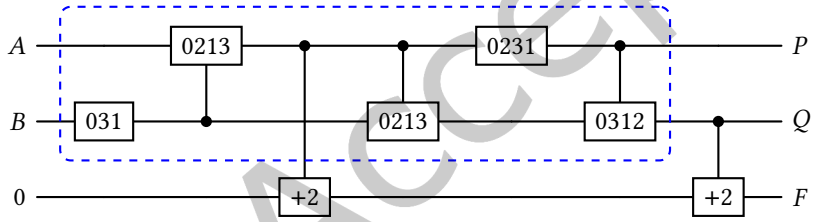


Fig. 8. Gate-level implementation of the 1-qudit greater-than comparator using the proposed genetic algorithm.

circuits with a fitness of 1.12. The distribution of results, shown in Figure 7, highlights the reproducibility of the GA in discovering the minimal greater-than comparator circuit, with minimal stochastic variation across runs.

The best-performing circuit achieved a length of 7 and a fitness of 1.142857, in its non-restoring form. The gate-level implementation of this solution for the greater-than comparator is illustrated in Figure 8. In the circuit, the inputs include qudits A, B, and a constant qudit initialized to 0. The outputs are P, Q, and F, where F presents the result of the comparison between A and B.

If input restoration is required, the circuit can be extended according to the general procedure described at the beginning of this section, by applying the inverse of the gates highlighted in the blue box in Figure 8. In this way, the total cost would be 12. In both restoring and non-restoring cases, the resulting designs remain substantially more efficient than its counterparts reported in [32, 66, 99], which require quantum costs of 25, 25, and 39, respectively.

These results demonstrate the effectiveness of the proposed genetic synthesis method in constructing optimized quaternary reversible circuits, achieving lower quantum cost and reducing the number of constant inputs and garbage outputs.

4.3 Equality Comparator

In this section, we present the equality comparator circuit (see Section 2.4.3) we synthesized and optimized with the GA-based framework described earlier in this paper. Across 20 independent GA runs, the best fitness values ranged between 1.05 and 1.142857, with correct circuit lengths varying from 7 to 20 gates. This variation reflects the stochastic nature of evolutionary search. Notably, all runs converged to functionally correct solutions (fitness > 1). Figure 9 summarizes the distribution of best fitness values across runs, showing consistent convergence close to the target truth table.

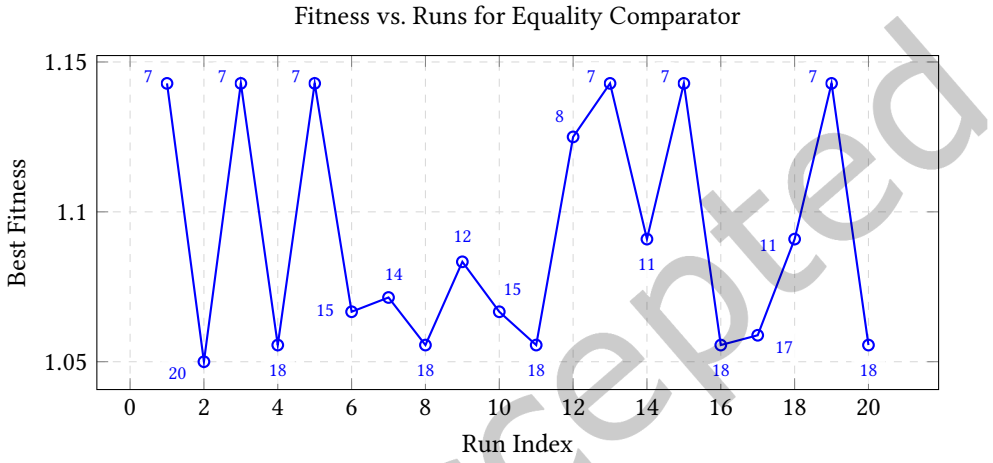


Fig. 9. Fitness across 20 GA runs for the 1-qudit equality comparator. Each point is the best circuit of a run; labels indicate circuit length.

The best solution identified for the equality comparator by our GA had a length of 7 and a fitness of 1.142857, and it is reported in Figure 10. As for the two previously described circuits, Figure 10 represents the equality comparator in its non-restoring form. This design minimizes resource usage while maintaining correct logical behavior. As shown in the circuit, the inputs consist of qudits A, B, and a constant input initialized to 0. The outputs are P, Q, and F, where F represents the target output that holds the result of the comparison between A and B. This compact design emerged after the GA explored longer candidate circuits, confirming the effectiveness of evolutionary search in discovering minimal implementations.

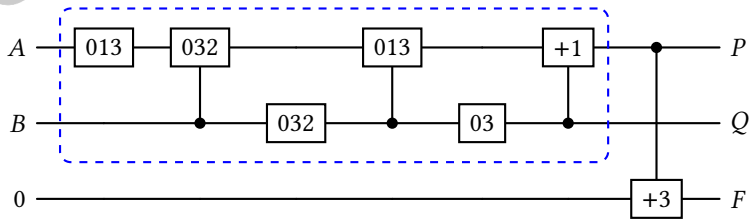


Fig. 10. Gate-level implementation of the 1-qudit equality comparator using the proposed genetic algorithm.

If input restoration is required, the circuit can be extended according to the general procedure described at the beginning of this section, by applying the inverse of the gates highlighted in the blue box in Figure 10. In that case, the total cost would be 13. The restoring version, therefore, matches the most efficient prior design [66] and still improves upon the others, which require costs of 15 and 15 in [32, 99].

4.4 1-qudit Full Comparator

In this section, we present the full comparator circuit (see Section 2.4.4) synthesized and optimized using the GA-based framework described earlier in this paper. This circuit serves as a fundamental building block for scalable N-qudit quaternary reversible comparators. The proposed design integrates the equality, less-than, and greater-than functions within a single structure, enabling efficient synthesis of comparative operations in the quaternary domain. As shown in Figure 11, the GA-based framework achieved best fitness values between 1.05 and 1.09 across 20 independent runs, with circuit lengths ranging from 11 to 20 gates. All runs successfully converged to valid and functionally correct circuits, demonstrating the robustness of the optimization process in producing compact, high-fitness designs with minimal circuit overhead.

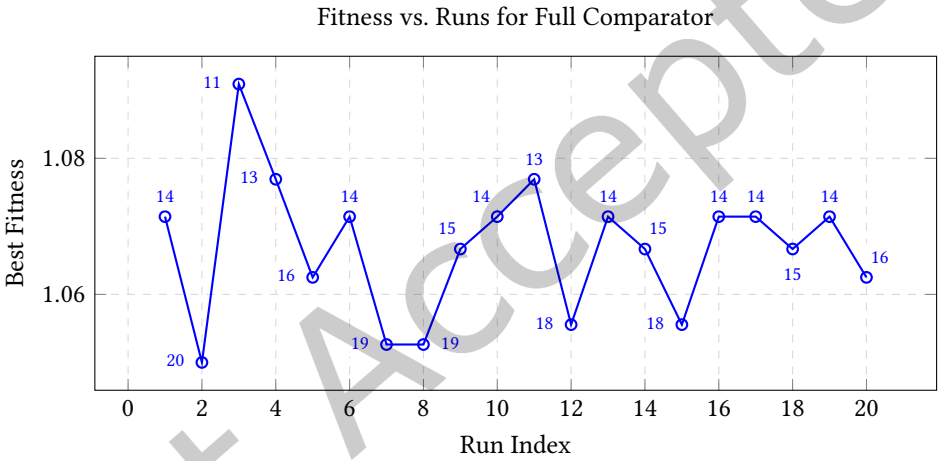


Fig. 11. Fitness values obtained across 20 GA runs for the 1-qudit full comparator. Each point represents the best circuit of a run; labels indicate circuit length.

The best design we synthesized by using the proposed approach achieved a quantum cost of 11, requiring only a single constant input and generating two garbage outputs. A detailed gate-level realization is depicted in Figure 12. As can be observed, inputs A and B serve as the primary data qudits, while the third input line is a constant qudit initialized to 0. On the output side, P and Q are garbage outputs, and F is the target output representing the result of the comparison. This circuit only employs four quaternary Shift and seven Muthukrishnan–Stroud (M–S) gates, maintaining both structural simplicity and functional completeness. A restoring form of this comparator can also be realized by reversing the computation on the blue box on the first two lines, resulting in a quantum cost of 17 while preserving the same functional behavior for the target output.

Compared to existing designs proposed in [32, 66, 99], the proposed full comparator offers a more efficient solution in terms of quantum cost, requiring only 11 gates in the non-restoring configuration and 17 in its restoring form. Both versions outperform previous implementations, whose reported costs are 27, 40, and 93, respectively.

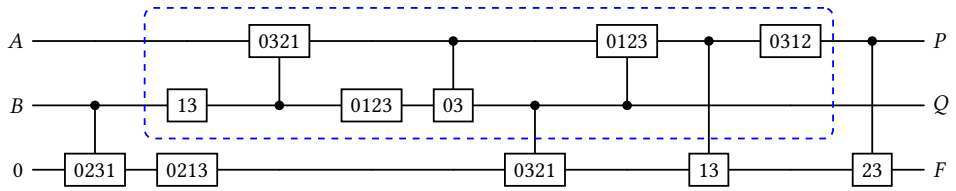


Fig. 12. Gate-level implementation of the 1-qudit full comparator circuit synthesized using the proposed genetic algorithm.

4.5 N-qudit Full Comparator

While in the previous section we presented the 1-qudit Full Comparator, we here discuss its generalization to N qudits. In principle, one may apply the same GA-based synthesis approach to N-qudits and synthesize the entire full comparator. However, in this section, we present how our approach can be used to synthesize and optimize all sub-modules of a circuit which are then merged. In particular, we can define an N-qudit full comparator by merging two building blocks, i.e., the 1-qudit full comparator (Section 2.4.4) and a sub-comparator circuit. These optimized modules can then be systematically composed to construct scalable N-qudit full comparator circuits without the need to reapply GA for a bigger circuit.

In a quaternary N-qudit comparator, two N-qudit numbers are compared through a two-phase process. First, each pair of corresponding qudits is compared in parallel using N separate 1-qudit comparators (see Section 4.4), generating intermediate comparison results. In the second phase, the outputs of these comparators are sequentially processed by $N-1$ sub-comparator circuits, starting from the least significant qudit, to derive the final comparison outcome between the two N-qudit numbers.

Table 5 presents the truth table for the sub-comparator circuit that we used as input for our GA to synthesize a circuit performing the intended functionality. Here, the output F encodes the quaternary comparison result between inputs F_0 and F_1 (outputs from previous steps). Only three outputs are valid: 1, 2, and 3, corresponding to less than, greater than, and equal to, respectively. Input pairs leading to undefined outcomes are denoted as 'x' (DC).

Across 20 independent GA runs, the best fitness values ranged between 1.1111 and 1.1667, with circuit lengths varying from 6 to 9 gates. This limited variation indicates that the GA consistently converged to compact and functionally correct designs, with most runs achieving near-optimal costs. Figure 13 summarizes the distribution of best fitness values across runs, showing that the majority of solutions reached the minimal circuit cost of 6 gates.

The optimal sub-comparator shown in Figure 14 and obtained through the proposed GA-based synthesis, in its non-restoring form, has a quantum cost of 6, requiring one constant input and producing two garbage outputs. The circuit takes three inputs: F_0, F_1, and a constant qudit initialized to 0. The outputs are R_0 and R_1 (garbage outputs), along with F (target output), which represents the comparison result. The implementation uses two Shift gate and four Muthukrishnan–Stroud (M–S) gates, resulting in a total quantum cost of 6.

As previously highlighted, the circuit in Figure 14 is in its non-restoring form. If input restoration is required, the circuit can be extended according to the general procedure described at the beginning of this section, by applying the inverse of the gates highlighted in the blue box in Figure 14, leading to a total cost of 9. In both cases, however, the total cost improves upon the others available in the literature, which require costs of 11 and 13 in [66, 99].

Using the 1-qudit full comparator (Section 4.4) and sub-comparator circuits as modules, we construct the scalable n-qudit comparator shown in Figure 15. The quantum cost ($QC_{\text{no-restore}}$ and QC_{restore}), number of constant

Table 5. Truth table of the sub-comparator circuit.

F_0	F_1	F
0	0	x
0	1	x
0	2	x
0	3	x
1	0	x
1	1	1
1	2	2
1	3	1
2	0	x
2	1	1
2	2	2
2	3	2
3	0	x
3	1	1
3	2	2
3	3	3

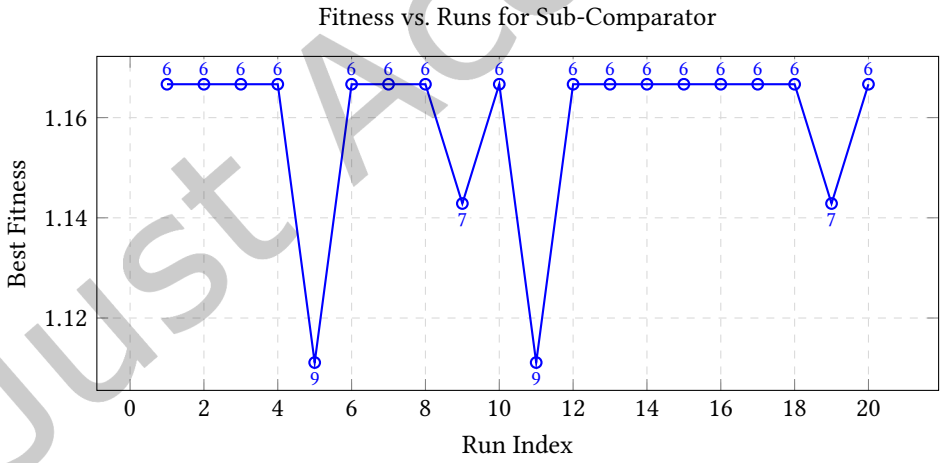


Fig. 13. Fitness values obtained across 20 GA runs for the Sub-Comparator. Each point represents the best circuit of a run; labels indicate circuit length.

inputs (N_{CI}), and garbage outputs (N_{GO}) for this comparator are given by Equations 4–7. It should be noted that if it is not essential to restore the inputs to their original states at the outputs, the non-restoring versions of the full comparator and sub-comparator circuits can be utilized. In this case, the total quantum cost of the proposed n-qudit

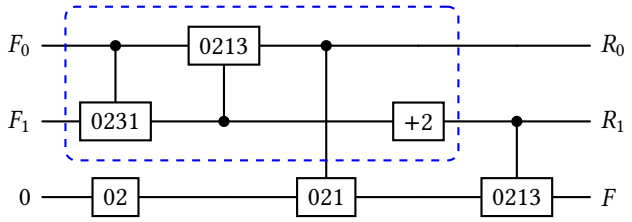


Fig. 14. Gate-level implementation of the Sub-Comparator circuit synthesized using the proposed genetic algorithm.

comparator is given by Equation 4. This configuration is suitable when the input states are not required for further computation or reuse, thereby reducing the overall gate count and improving efficiency. Conversely, if input restoration is required, the restoring versions of the full comparator and sub-comparator circuits are employed. Under this condition, additional gate are needed to return all input lines to their initial values. Consequently, the total quantum cost increases to $26N - 9$, as shown in Equation 5. In both restoring and non-restoring implementations, the number of constant inputs and garbage outputs remains unchanged—specifically, $2N - 1$ constant inputs and $4N - 2$ garbage outputs, as shown in Equations 6 and 7.

$$QC_{\text{no-restore}} = 17N - 6, \tag{4}$$

$$QC_{\text{restore}} = 26N - 9, \tag{5}$$

$$N_{\text{CI}} = 2N - 1, \tag{6}$$

$$N_{\text{GO}} = 4N - 2, \tag{7}$$

As an illustration, consider two 2-qudit numbers A and B in quaternary representation. To perform the comparison between A and B, the initial step involves comparing A_0 with B_0, and A_1 with B_1, respectively, using the proposed 1-qudit comparator. In the next step, the outputs of these individual comparisons are evaluated using the designed sub-comparator circuit. The final comparison result is determined based on Table 5 and the logic described in Equations 8, 9, and 10:

$$A_1A_0 = B_1B_0 \quad \text{if} \quad A_1 = B_1 \quad \text{and} \quad A_0 = B_0 \tag{8}$$

$$A_1A_0 > B_1B_0 \quad \text{if} \quad A_1 > B_1 \quad \text{or} \quad A_1 = B_1 \quad \text{and} \quad A_0 > B_0 \tag{9}$$

$$A_1A_0 < B_1B_0 \quad \text{if} \quad A_1 < B_1 \quad \text{or} \quad A_1 = B_1 \quad \text{and} \quad A_0 < B_0 \tag{10}$$

Based on Equation 8, the output is set to 3 if both comparisons in the first step return 3, indicating equality. As described in Equation 9, the output becomes 2 when the second comparator yields 2, or when the second comparator gives 3 and the first gives 2. Similarly, from Equation 10, the result is 1 if the second comparator is equal to 1, or if the second returns 3 and the first returns 1.

The realization of the previously discussed quaternary 2-qudit comparator circuit is illustrated in Figure 16. As shown in Figure 16a, the design follows a hierarchical structure that integrates two GA-optimized 1-qudit full comparator circuits with a single GA-optimized sub-comparator block.

The first stage consists of two parallel full comparators: The top unit processes the least significant qudits A_0 and B_0, while the second unit evaluates the most significant qudits A_1 and B_1. Each of these blocks produces a set of outputs, namely the target comparison result R_i (the third output line of each full comparator) along with

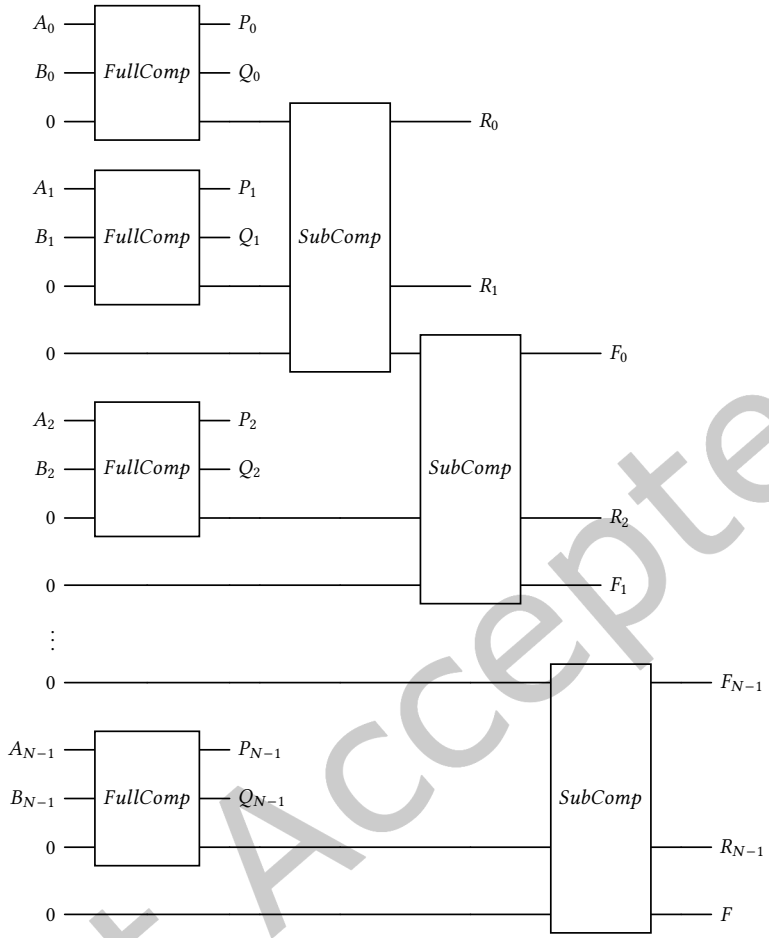
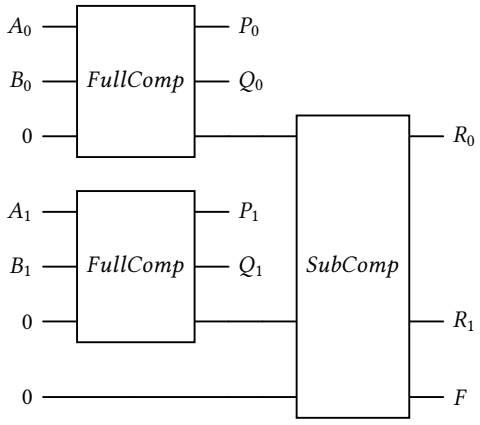


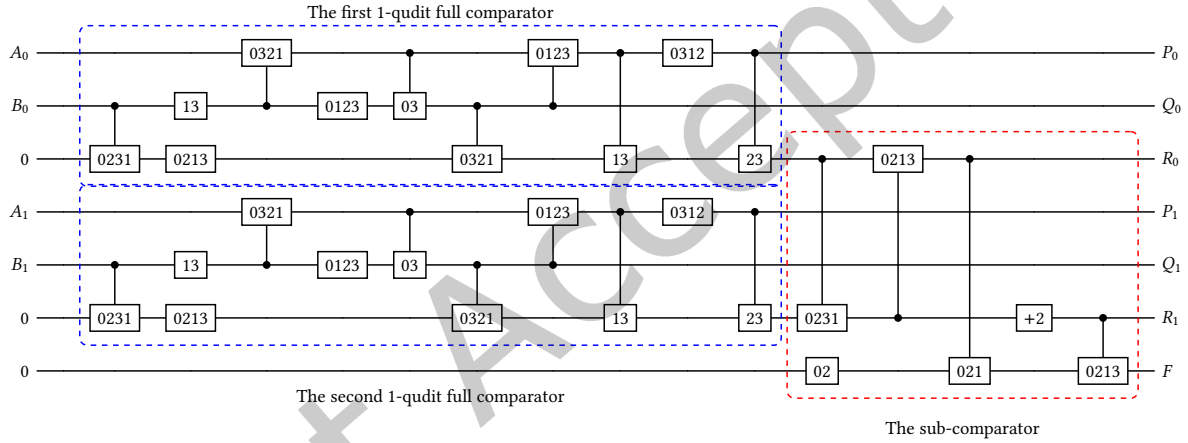
Fig. 15. The proposed quaternary reversible n-qudit comparator circuit constructed from GA-synthesized and optimized building blocks.

auxiliary garbage outputs P_i and Q_i. These intermediate values encode whether the respective qudits are equal, greater, or lower in the first stage of the comparison. The second stage uses the sub-comparator to combine the results from the two 1-qudit comparators and generate the final comparison outcome F. This hierarchical decision process guarantees correctness and scalability, since only one additional sub-comparator is needed per qudit level.

Figure 16b illustrates the full gate-level realization of the 2-qudit comparator using Shift and Muthukrishnan–Stroud (M–S) gates. The dashed blue boxes correspond to the two 1-qudit full comparator circuits, while the dashed red block marks the sub-comparator unit. The design uses a total of 28 gates, including 18 quaternary M–S gates and 10 quaternary Shift gates, leading to a quantum cost that matches the prediction of Equation 4 for $n = 2$. In addition, this 2-qudit circuit requires three constant inputs, which conforms to Equation 6, and produces six garbage outputs, in agreement with Equation 7.



(a) Structure using full and sub comparator blocks.



(b) Realization using Shift and M–S gates.

Fig. 16. Logical implementation of the quaternary reversible 2-qudit comparator: (a) structure using full and sub comparator blocks, and (b) realization using Shift and M–S gates.

Example 1. *Let's consider the case where* $A = (A_1A_0) = (21)$ *and* $B = (B_1B_0) = (23)$. *The first full comparator evaluates* $A_0 = 1$ *against* $B_0 = 3$, *resulting in* $F_0 = 1$ *since* $A_0 < B_0$. *The second comparator simultaneously checks* $A_1 = 2$ *against* $B_1 = 2$, *producing* $F_1 = 3$ *since the qudits are equal. These outputs are then passed to the sub-comparator. Because* $F_1 = 3$, *the sub-comparator forwards the result of* F_0, *which is 1. Therefore, the final output of the circuit is* $F = 1$, *indicating that* $A < B$. *This matches the intended quaternary comparison logic described in Equation 10.*

This example confirms that the proposed architecture correctly evaluates multi-qudit inputs by prioritizing the most significant qudits while deferring to less significant ones only in cases of equality. More generally, the

2-qudit design demonstrates how the GA-synthesized building blocks can be composed into larger comparators, thereby ensuring both scalability and correctness.

5 Evaluation and Discussion

This section presents a structured evaluation of the proposed quaternary comparator designs. The analysis focuses on three key performance metrics [26, 57]: quantum cost, constant inputs, and garbage outputs, as explained earlier in Section 1.

We begin by evaluating the basic building blocks of the 1-qudit comparators: the less-than, greater-than, equality comparators and full comparator, along with the sub-comparator implemented using quaternary logic previously discussed in Section 4. In our evaluation, we consider both the quantum cost of circuits with input restoration (i.e., mirrored circuits), $\mathbf{QC}_{\text{restore}}$, and the cost of circuits without input restorationm, $\mathbf{QC}_{\text{no-restore}}$. Then, we examine the resource efficiency of the proposed N-qudit comparator by comparing it with other existing implementations (Section 5.2). Throughout, performance improvements are quantified and discussed with reference to previous works [32, 66, 99].

5.1 Evaluation of 1-qudit comparator circuits

As shown in Table 6, the proposed 1-qudit less-than comparator demonstrates substantial improvements over existing designs across all key metrics. The proposed circuit obtained through our GA achieves a quantum cost of 12 with input restoration and 7 without input restoration. Compared to the best existing design by Norouzi et al. [66], which requires a cost of 22, it represents quantum cost reductions of 45.45% and 68.18% for the restoring and non-restoring configurations, respectively. Furthermore, the proposed design uses only 1 constant input, whereas all other designs require 2, reflecting a more efficient initialization process. In terms of garbage outputs, the proposed comparator produces only 2 outputs, compared to 3 in all prior works.

Overall, the proposed design offers a significantly more optimized solution, reducing computational overhead, while minimizing ancillary resources. These results emphasize its effectiveness for scalable and efficient reversible quaternary logic circuit implementations, as well as the effectiveness of the proposed GA-based synthesis and optimization process.

Table 6. Comparison between proposed 1-qudit less-than comparator and existing designs.

Design	$\mathbf{QC}_{\text{restore}}$	$\mathbf{QC}_{\text{no-restore}}$	$\mathbf{N}_{\text{CI}}$	$\mathbf{N}_{\text{GO}}$
Proposed Design	12	7	1	2
[66]	22	N/A	2	3
[99]	25	N/A	2	3
[32]	39	N/A	2	3

Table 7 highlights the key metrics of the proposed 1-qudit greater-than comparator with respect to existing designs. The circuit based on the proposed GA achieves a quantum cost of 12 with input restoration and 7 without input restoration . Although including the mirrored version increases the cost by approximately 71% compared to the non-mirrored case, it still remains lower than that of the most efficient existing designs by Norouzi et al. [66] and Zhang et al. [99], which require a cost of 25. This represents quantum cost reductions of 52% and 72% for the restoring and non-restoring configurations, respectively. The proposed design also requires only 1 constant input, which is 50% less than the number used in previous approaches. Furthermore, it produces 2 garbage outputs, offering an improvement over the 3 outputs generated by all existing methods. The reductions

in quantum cost, constant inputs, and garbage outputs collectively enhance the practicality and efficiency of the comparator, making it a strong candidate for implementation in scalable quaternary quantum systems.

Table 7. Comparison between proposed 1-qudit greater-than comparator and existing designs.

Design	$\mathbf{QC}_{\text{restore}}$	$\mathbf{QC}_{\text{no-restore}}$	$\mathbf{N}_{\text{CI}}$	$\mathbf{N}_{\text{GO}}$
Proposed Design	12	7	1	2
[66]	25	N/A	2	3
[99]	25	N/A	2	3
[32]	39	N/A	2	3

Table 8 presents a comparison between the proposed 1-qudit equality comparator and previously reported designs in [32, 66, 99]. The proposed circuit achieves a quantum cost of 13 with input restoration and 7 without input restoration. While the restoring configuration achieves the same quantum cost as the most efficient prior design by Norouzi et al. [66], it still improves upon the designs of Zhang et al. [99] and Khan et al. [32], which each exhibit a quantum cost of 15. The non-restoring realization further reduces the quantum cost by approximately 46.15% relative to the best configuration, offering a lower-cost alternative when input restoration is unnecessary. In terms of ancillary resources, all compared designs utilize one constant input and produce two garbage outputs, except for the design in [32], which requires two constants and three garbage outputs.

Table 8. Comparison between proposed 1-qudit equality comparator and existing designs.

Design	$\mathbf{QC}_{\text{restore}}$	$\mathbf{QC}_{\text{no-restore}}$	$\mathbf{N}_{\text{CI}}$	$\mathbf{N}_{\text{GO}}$
Proposed Design	13	7	1	2
[66]	13	N/A	1	2
[99]	15	N/A	1	2
[32]	15	N/A	2	3

Table 9 presents a comparative analysis of the proposed 1-qudit full comparator circuit against existing designs in terms of quantum cost, constant inputs, and garbage outputs. The circuit we obtained through GA achieves a quantum cost of 17 in the restoring configuration and 11 in the non-restoring configuration. Consequently, when considering the mirrored version, the quantum cost increases by approximately 55% compared to the non-mirrored configuration. Compared to the most efficient existing design by Norouzi et al. [66] with a cost of 27, the proposed comparator achieves reductions of 37% and 59% for the restoring and non-restoring configurations, respectively. Even greater improvements are observed over the designs of Zhang et al. [99] and Khan et al. [32], with quantum cost reductions of up to 57% and 81%, respectively. Moreover, the design we proposed maintains the lowest number of constant inputs (1) and garbage outputs (2), matching the best in class while outperforming others that require up to 6 constant inputs and 9 garbage outputs. These results clearly demonstrate the efficiency and optimization of the proposed design over existing approaches.

Table 10 highlights the key metrics of the proposed 1-qudit sub-comparator circuit with respect to existing designs. Achieving a quantum cost of just 9 in the restoring configuration and 6 in the non-restoring configuration, resulting in a 45.45% reduction in quantum cost compared to the design in [99] and a 53.85% reduction relative to [66]. Concerning ancillary resource requirements, the proposed design utilizes only one constant input and generates two garbage outputs, thereby matching the efficiency of the most optimized prior design [99], while

Table 9. Comparison of the proposed 1-qudit full comparator with existing designs.

Design	$\mathbf{QC}_{\text{restore}}$	$\mathbf{QC}_{\text{no-restore}}$	$\mathbf{N}_{\text{CI}}$	$\mathbf{N}_{\text{GO}}$
Proposed Design	17	11	1	2
[66]	27	N/A	1	2
[99]	40	N/A	2	3
[32]	93	N/A	6	9

offering a significant improvement over the design in [66], which needs two constant inputs and produces three garbage outputs.

Table 10. Comparison of the proposed 1-qudit sub-comparator with existing designs.

Design	$\mathbf{QC}_{\text{restore}}$	$\mathbf{QC}_{\text{no-restore}}$	$\mathbf{N}_{\text{CI}}$	$\mathbf{N}_{\text{GO}}$
Proposed Design	9	6	1	2
[99]	11	N/A	1	2
[66]	13	N/A	2	3

5.2 Evaluation of the N-qudit comparator circuit

Table 11 presents detailed evaluation of our proposed quaternary N-qudit comparator circuits with respect to previously reported designs [32, 66, 99]. As shown in the table, the proposed circuit achieves a quantum cost of $26N - 9$ in the restoring configuration and $17N - 6$ in the non-restoring configuration, both of which are significantly lower than those of existing approaches. For instance, the design in [66] exhibits a quantum cost of $40N - 13$, whereas the designs in [99] and [32] require $51N - 11$ and $246N - 60$, respectively. In addition, the proposed comparator uses only $2N - 1$ constant inputs and generates $4N - 2$ garbage outputs, compared to $3N - 1$ and $5N - 2$ in [99], $3N - 2$ and $5N - 3$ in [66], and $13N - 4$ and $27N - 6$ in [32]. As a result, our N-qudit comparator demonstrates superior efficiency due to its lower quantum cost, constant inputs, and garbage outputs. Since a lower value of these important parameters directly correlates with higher efficiency in quantum circuit design, this reinforces the advantage of our approach.

Table 11. Comparison between the proposed N-qudit comparator and existing designs.

Design	$\mathbf{QC}_{\text{restore}}$	$\mathbf{QC}_{\text{no-restore}}$	$\mathbf{N}_{\text{CI}}$	$\mathbf{N}_{\text{GO}}$
Proposed Design	$26N - 9$	$17N - 6$	$2N - 1$	$4N - 2$
[99]	$51N - 11$	N/A	$3N - 1$	$5N - 2$
[66]	$40N - 13$	N/A	$3N - 2$	$5N - 3$
[32]	$246N - 60$	N/A	$13N - 4$	$27N - 6$

6 Related work

Synthesis and optimization of quantum circuits are pivotal for translating algorithms into efficient, executable programs on current hardware, yet these steps are still largely carried out through manual [6, 20, 85], heuristic-driven effort in practice, motivating automated approaches in this space [96].

Evolutionary computation and genetic algorithms (GAs) have been extensively applied to the synthesis and optimization of reversible and quantum circuits. Early work by Lukac et al. [47] demonstrated that evolutionary techniques could effectively address reversible circuit synthesis, paving the way for GA-based approaches in quantum logic design. This direction was further advanced by Mukherjee et al. [60], who showed that GA can synthesize quantum circuits for both completely and incompletely specified functions, often achieving better cost efficiency than analytical methods. Later studies refined these methods by introducing structural restrictions for ternary quantum spaces [48], exploring high-speed GA implementations with low-level parallelization [45], and proposing frameworks such as RIMEP2 for evolutionary circuit generation [25]. Datta et al. [12] further enhanced GA-based synthesis by incorporating output permutation strategies. Ruican et al. [78] demonstrated that GA can successfully synthesize circuits up to 7 qubits using adaptive parameter control, highlighting its scalability to larger problem sizes. Complementing these domain-specific studies, Zebulum et al. [98] highlighted the generality of evolutionary design for electronic systems, underscoring the adaptability of GA to quantum circuit optimization. More recently, Sarvaghad-Moghaddam et al. [81] proposed a multi-objective GA framework for quantum circuits, and Islam et al. [28] introduced a fuzzy-logic–assisted GA that further improved convergence and solution quality, demonstrating the continued evolution and adaptability of GA-based synthesis techniques. Finally, mutation techniques in GA-based binary quantum circuit synthesis are evaluated in [41]. These contributions demonstrate the continued evolution and adaptability of GA-based synthesis techniques, while still being limited to binary logic.

The application of GA has also been extended to multi-valued systems. Khan and Perkowski presented a GA-based approach for the synthesis of multi-output ternary reversible functions [33], demonstrating the applicability of evolutionary methods to multi-valued quantum logic. Khanom et al. presented another GA-based framework for the synthesis of ternary reversible circuits, highlighting the potential of evolutionary computation in multi-valued logic [38]. Deibuk et al. designed a ternary reversible adder using GA [15], achieving significant reductions in gate count compared to traditional synthesis. Subsequently, GA was employed for ternary comparator synthesis [14], producing efficient realizations of equality and magnitude comparison circuits, which are inherently based on ternary numerical systems (Galois Field 3). Hybrid GA-based approaches have also been proposed to further improve synthesis efficiency in ternary reversible circuits [30] and for the synthesis of a quantum dot ternary full-adder [39]. These works demonstrated the scalability of GA to higher-radix systems, though they were limited to ternary logic. Conversely, our work is based on quaternary logic and in our GA we implement mutations specifically targeting quaternary gates. Only a few works in the literature target the automatic synthesis of quaternary circuits, but none of them applied a GA-based process as we propose in this paper. For example, [36] proposed a framework to automatically synthesize and minimize quaternary circuits using pseudo-Kronecker Galois field decision diagram (QGFDD). However, the results we obtained with our proposed approach are better than those reported in [36], indicating the flexibility and effectiveness of GA in quantum circuit synthesis.

In parallel, many analytical approaches have addressed comparators. Several works focus on designing efficient versions of comparators, with most of them focusing on more efficient design [72, 77, 80], while others focusing on introducing new gate implementations, e.g., using the decomposition of n-bit Toffoli gates by symmetrical logical structures and adjustable support qubits [11]. A direct comparison with these comparators is not feasible, as they are designed for binary logic, whereas our approach targets quaternary comparators. Khan [32] presented the first systematic designs for quaternary reversible comparators using Muthukrishnan–Stroud (M–S) gates. Although functionally correct, these circuits were not optimized in terms of cost or garbage outputs. Later, Norouzi et al. [66] provided a comparative study of quaternary reversible logic circuits, including gate libraries and small-scale designs, establishing baseline performance metrics, but without specifically targeting comparators. More recently, Zhang et al. [99] introduced optimized multi-valued comparator circuits by extending Toffoli-like gates to higher-radix systems. Their work improved quantum cost and hardware efficiency, but the designs were derived analytically and did not exploit heuristic search, as we propose in this paper. At the same time,

quaternary comparators have primarily been constructed using analytical methods [32, 66, 99], often leaving room for further cost reduction and scalability improvements. To the best of our knowledge, no prior work has applied GA to the synthesis of quaternary comparator circuits. Our work addresses this gap by employing GA to synthesize compact quaternary reversible comparators, aiming to minimize quantum cost, garbage outputs, and constant inputs, while enabling scalable N-qudit comparison.

Several approaches to multiple-valued and quaternary reversible circuit synthesis adopt different underlying algebraic models. In this work, we rely on the finite field GF(4), which enables a compact formulation of reversible transformations through linear and affine operations, as commonly done in multiple-valued quantum logic frameworks based on Galois fields [50]. This choice is also consistent with the general theory of multiple-valued logic representations, where field-based encodings provide advantageous algebraic properties for synthesis and optimization [55]. However, alternative numerical systems have been explored in the literature. In particular, quaternary logic can also be modeled over the ring $\mathbb{Z}_4$, which offers a more direct arithmetic interpretation but lacks the full structure of a field, thus limiting the applicability of linear decomposition techniques typically used in GF(4)-based synthesis [29]. Similarly, permutation- and decomposition-based approaches treat reversible functions independently of the chosen numerical representation [70], while decision-diagram-based techniques such as QMDDs enable scalable synthesis of quantum circuits without assuming a particular encoding of logic values [93].

7 Threats to validity

In this section, we discuss potential threats to the validity [79] of our work and the actions we have taken to mitigate them.

Internal validity refers to the fact that the different outcomes obtained with the analyzed techniques and tools are actually caused by the different approaches themselves and by the way the experiments were carried out, and not by methodological errors. To mitigate this risk, we have carefully checked the code executed in our experiments, as well as the one implementing the GA, to see if there could be other factors that may have caused the outcome, such as errors in the tools or in the experimental code we wrote. Moreover, in our approach, we rely on a well-known and widely used library, namely jMetalPy [5]. GAs inherently possess randomness because the initial population is generated randomly, and mutations and crossover are applied based on their probabilities. Consequently, we conducted 20 runs of each experiment. For certain circuits, we only obtained the best circuit a few times. Running more experiments or increasing the number of iterations for each run could improve this result.

A possible threat to the *construct validity* comes from the assumption that the measures we considered in our fitness function are suitable to correctly assess the quantum circuits under synthesis. On the one hand, for what concerns the number of errors, including this factor in a fitness function is trivial, as we would like to obtain a behaviorally correct circuit. On the other hand, our fitness considers the cost of the circuit being synthesized, as we would like to obtain the most optimized, i.e., the most compact, circuit. We rely on the literature for this, where similar measures are often used to assess the cost of quantum circuits [3, 26, 32, 43, 57, 66, 99]. However, we did not include the number of constant inputs and garbage outputs in the fitness function, as these parameters are largely constrained by the reversible circuit model and the chosen encoding: Our algorithm starts with an already defined number of input and output qudits, and no mutation changes them. Nevertheless, we evaluated them separately in the experimental analysis (Section 5), where we explicitly compared our designs with existing approaches. Similarly, when it comes to M-S gate cost, we did not consider technology-dependent cost model, such as the Maslov costs for binary circuits [52]. Instead, we adopted the commonly used abstraction in quaternary logic synthesis, where such gates are treated as unit-cost primitives to enable fair and consistent comparison

with existing designs. Exploring more accurate, technology-dependent cost models is an interesting direction for future work.

External validity is concerned with whether we can generalize the results outside the scope of the presented experiments. We conducted several preliminary experiments to determine the optimal values for each algorithm parameter and assess their impact on the results. However, we can not guarantee that different parameter values would lead to different outcomes. Furthermore, for what concerns the case studies we analyzed, we tried to include different circuits having different truth tables. We believe that the considered target functions are representative enough of possible quantum circuits and that our GA-based synthesis and optimization approach could be generalized to any other circuit, but further experiments may be needed to generalize our results and prove that the proposed GA could work on any quaternary reversible circuit. Finally, in this work, we assume a uniform cost for controlled gates, which is a common abstraction adopted in the literature on multi-valued quantum circuits. We note, however, that in practical quantum computing platforms the implementation cost of controlled operations can vary significantly depending on the underlying technology and physical realization. As a consequence, the cost evaluation in the proposed fitness function should be interpreted as technology-agnostic and primarily intended for relative comparison with existing works. Investigating technology-aware cost models and their impact on the proposed approach is an interesting direction for future work.

8 Conclusion

In this work, we introduced a genetic algorithm-based methodology for synthesizing quaternary reversible comparator circuits. This approach is complemented by a dedicated set of mutations that enable the generation of both functionally correct and optimized quaternary reversible circuits. Our GA-based synthesis and optimization approach employs a fitness function that considers both the correctness of a circuit and the number of gates. This approach aims to minimize the quantum cost associated with the circuit. During the circuit evolution, our algorithm monitors for stagnation and introduces diversity to prevent the algorithm from getting stuck in local optima. Additionally, the algorithm performs mutations directly targeting the structure of the circuit and allows for crossover operations.

We demonstrated the functionality and generalizability of our approach by synthesizing and optimizing quaternary reversible 1-qudit comparators, including the less-than, greater-than, equality, and full comparator. Furthermore, we obtained all the sub-components necessary for defining a N-qudit comparator. Experimental evaluations on all the analyzed circuits revealed a significant reduction in quantum cost compared to existing methods. Moreover, our approach effectively minimizes garbage outputs and ancilla usage. By bridging evolutionary synthesis with structural optimization, the presented approach contributes a practical step toward building more compact, error-resilient, and hardware-friendly quantum circuits.

As future work, we plan to extend the framework to accept algebraic or symbolic function representations as input, thereby overcoming the scalability limitations of truth-table-based synthesis. Future work will also focus on extending this methodology to additional quaternary arithmetic operations and integrating the optimizer into high-level quantum compilation flows. For example, we are working on integrating the GA-based optimization phase into the QuTiP-MRL library [73] for multiple-valued reversible logic simulation. This addition would allow users to seamlessly design a circuit and obtain an equivalent and optimized version.

Acknowledgements

This work has been supported by the National Recovery and Resilience Plan (NRRP), Mission 4, Component 2, Investment 1.1, under Call No. 104 (2.2.2022) by the Italian Ministry of University and Research (MUR), funded by the EU – NextGenerationEU – SAFEST project (CUP F53D23004230006), Grant Decree n. 861 (16-6-2023) by MUR. This work was supported in part by project SERICS (PE00000014) under the NRRP MUR program funded by the

EU - NGEU. The work of Andrea Bombarda has been supported by PNRR - ANTHEM (AdvaNced Technologies for Human-centrEd Medicine) - Grant PNC0000003 – CUP: B53C22006700001 - Spoke 1 - Pilot 1.4. Views and opinions expressed are however those of the authors only and do not necessarily reflect those of the European Union or the Italian MUR. Neither the European Union nor the Italian MUR can be held responsible for them. Valentina Ciriani is a member of the Gruppo Nazionale Calcolo Scientifico-Istituto Nazionale di Alta Matematica (GNCS-INdAM), which provided partial support for this work.

References

[1] Jing-wen Ai, Ting-yan Zhang, Si-jun Lu, Ming-qiang Bai, and Zhi-wen Mo. 2025. Design and optimization of the multi-valued quantum adder. *Physica Scripta* 100, 4 (2025), 045115.

[2] Marco Anderlini, Patricia J Lee, Benjamin L Brown, Jennifer Sebby-Strabley, William D Phillips, and James V Porto. 2007. Controlled exchange interaction between pairs of neutral atoms in an optical lattice. *Nature* 448, 7152 (2007), 452–456.

[3] Adriano Barenco, Charles H Bennett, Richard Cleve, David P DiVincenzo, Norman Margolus, Peter Shor, Tycho Sleator, John A Smolin, and Harald Weinfurter. 1995. Elementary gates for quantum computation. *Physical review A* 52, 5 (1995), 3457.

[4] Charles H Bennett. 1973. Logical reversibility of computation. *IBM journal of Research and Development* 17, 6 (1973), 525–532.

[5] Antonio Benítez-Hidalgo, Antonio J. Nebro, José García-Nieto, Izaskun Oregi, and Javier Del Ser. 2019. jMetalPy: A Python framework for multi-objective optimization with metaheuristics. *Swarm and Evolutionary Computation* 51 (2019), 100598. doi:10.1016/j.swevo.2019.100598

[6] Anna Bernasconi, Valentina Ciriani, Asma Taheri Monfared, and Stefano Zanoni. 2023. Compact Quantum Circuits for Dimension Reducible Functions. In *2023 26th Euromicro Conference on Digital System Design (DSD)*. 776–781. doi:10.1109/DSD60849.2023.00111

[7] Dimitris Bertsimas and John Tsitsiklis. 1993. Simulated Annealing. *Statist. Sci.* 8, 1 (Feb. 1993). doi:10.1214/ss/1177011077

[8] Dinabandhu Bhandari, Nikhil R. Pal, and Sankar K. Pal. 1994. Directed mutation in genetic algorithms. *Information Sciences* 79, 3-4 (1994), 251–270. doi:10.1016/0020-0255(94)90123-6

[9] Kenneth R. Brown, John Chiaverini, Jeremy M. Sage, and Hartmut Häffner. 2021. Materials challenges for trapped-ion quantum computers. *Nature Reviews Materials* 6, 10 (March 2021), 892–905. doi:10.1038/s41578-021-00292-1

[10] Dusanka Bundalo, Zlatko Bundalo, and Branimir Dordjevic. 2005. Design of quaternary logic systems and circuits. *Facta universitatis - series: Electronics and Energetics* 18, 1 (2005), 45–56. doi:10.2298/fuee0501045b

[11] Shanyan Chen, Ali Al-Bayaty, Xiaoyu Song, and Marek Perkowski. 2025. Stesso: A reconfigurable decomposition of n-bit Toffoli gates using symmetrical logical structures and adjustable support qubits. arXiv:2510.26116 [quant-ph] https://arxiv.org/abs/2510.26116

[12] Kamalika Datta, Indranil Sengupta, Hafizur Rahaman, and Rolf Drechsler. 2013. An evolutionary approach to reversible logic synthesis using output permutation. In *2013 8th IEEE Design and Test Symposium*. IEEE, 1–6.

[13] Kalyanmoy Deb and Tushar Goel. 2001. *Controlled Elitist Non-dominated Sorting Genetic Algorithms for Better Convergence*. Springer Berlin Heidelberg, 67–81. doi:10.1007/3-540-44719-9_5

[14] Vitaly Deibuk and Andrij Biloshytskyi. 2015. Genetic synthesis of new reversible/quantum ternary comparator. *Advances in Electrical and Computer Engineering* 15, 3 (2015), 147–152.

[15] Vitaly G Deibuk and Andrij V Biloshytskyi. 2015. Design of a ternary reversible/quantum adder using genetic algorithm. *International Journal of Information Technology and Computer Science* 7, 9 (2015), 38–45.

[16] J. Eisert, K. Jacobs, P. Papadopoulos, and M. B. Plenio. 2000. Optimal local implementation of nonlocal quantum gates. *Physical Review A* 62, 5 (Oct. 2000). doi:10.1103/physreva.62.052317

[17] Ehsan Faghih, MohammadReza Taheri, Keivan Navi, and Nader Bagherzadeh. 2023. Efficient realization of quantum balanced ternary reversible multiplier building blocks: A great step towards sustainable computing. *Sustainable Computing: Informatics and Systems* 40 (2023), 100908.

[18] Arkady Fedorov, Lars Steffen, Matthias Baur, Marcus P da Silva, and Andreas Wallraff. 2012. Implementation of a Toffoli gate with superconducting circuits. *Nature* 481, 7380 (2012), 170–172.

[19] Wei-Bo Gao, Ping Xu, Xing-Can Yao, Otfried Gühne, Adán Cabello, Chao-Yang Lu, Cheng-Zhi Peng, Zeng-Bing Chen, and Jian-Wei Pan. 2010. Experimental realization of a controlled-NOT gate with four-photon six-qubit cluster states. *Physical review letters* 104, 2 (2010), 020501.

[20] Yan Ge, Wu Wenjie, Chen Yuheng, Pan Kaisen, Lu Xudong, Zhou Zixiang, Wang Yuhan, Wang Ruocheng, and Yan Junchi. 2024. Quantum Circuit Synthesis and Compilation Optimization: Overview and Prospects. doi:10.48550/ARXIV.2407.00736

[21] Farzad Ghannadian, Cecil Alford, and Ron Shonkwiler. 1996. Application of random restart to genetic algorithms. *Information Sciences* 95, 1-2 (Nov. 1996), 81–102. doi:10.1016/s0020-0255(96)00121-1

[22] S Girija and BG Sangeetha. 2021. An Insight into the Existing Reversible Arithmetic and Logic Unit Designs. In *Proceeding of Fifth International Conference on Microelectronics, Computing and Communication Systems: MCCS 2020*. Springer, 383–400.

[23] Andrew D Greentree, Simon G Schirmer, F Green, Lloyd CL Hollenberg, AR Hamilton, and RG Clark. 2004. Maximizing the Hilbert space for a finite number of distinguishable quantum states. *Physical review letters* 92, 9 (2004), 097901.
[24] Haoran Gu, Handing Wang, Cheng He, Bo Yuan, and Yaochu Jin. 2025. Large-scale multiobjective evolutionary algorithm guided by low-dimensional surrogates of scalarization functions. *Evolutionary Computation* 33, 3 (2025), 309–334.
[25] Fatima Zohra Hadjam and Claudio Moraga. 2014. RIMEP2: Evolutionary design of reversible digital circuits. *ACM Journal on Emerging Technologies in Computing Systems (JETC)* 11, 3 (2014), 1–23.
[26] Majid Haghparast and Asma Taheri Monfared. 2017. Novel quaternary quantum decoder, multiplexer and demultiplexer circuits. *International Journal of Theoretical Physics* 56, 5 (2017), 1694–1707.
[27] Majid Haghparast and Asma Taheri Monfared. 2018. Designing novel quaternary quantum reversible subtractor circuits. *International Journal of Theoretical Physics* 57, 1 (2018), 226–237.
[28] Ishraq Islam, Vinayak Jha, Sneha Thomas, Kieran F Egan, Alvir Nobel, Serom Kim, Manu Chaudhary, Sunday Ogundele, Dylan Kneidel, Ben Phillips, et al. 2025. Quantum Circuit Synthesis Using Fuzzy-Logic-Assisted Genetic Algorithms. *Algorithms* 18, 4 (2025), 178.
[29] Zoubida Jadda and Patrice Parraud. 2010. Z4-Nonlinearity of a Constructed Quaternary Cryptographic Functions Class. In *Sequences and Their Applications – SETA 2010*, Claude Carlet and Alexander Pott (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 270–283.
[30] Musharrat Khan and Jacqueline E Rice. 2019. Hybrid GA Synthesis of Ternary Reversible Circuits Using Max-Min Algebra. *Journal of Multiple-Valued Logic & Soft Computing* 32 (2019).
[31] Mozammel HA Khan. 2008. A recursive method for synthesizing quantum/reversible quaternary parallel adder/subtractor with look-ahead carry. *Journal of Systems Architecture* 54, 12 (2008), 1113–1121.
[32] Mozammel HA Khan. 2008. Synthesis of quaternary reversible/quantum comparators. *Journal of Systems Architecture* 54, 10 (2008), 977–982.
[33] Mozammel HA Khan and Marek Perkowski. 2004. Genetic algorithm based synthesis of multi-output ternary functions using quantum cascade of generalized ternary gates. In *Proceedings of the 2004 Congress on Evolutionary Computation (IEEE Cat. No. 04TH8753)*, Vol. 2. IEEE, 2194–2201.
[34] Mozammel HA Khan and Marek A Perkowski. 2007. Quantum ternary parallel adder/subtractor with partially-look-ahead carry. *Journal of Systems Architecture* 53, 7 (2007), 453–464.
[35] M. H. A. Khan and M. A. Perkowski. 2007. GF(4) based synthesis of quaternary reversible/quantum logic circuits. In *Proceedings of the 37th International Symposium on Multiple-Valued Logic (ISMVL)*. 11.
[36] Mozammel H. A. Khan, Himanshu Thapliyal, and Edgard Munoz-Coreas. 2016. Automatic synthesis of quaternary quantum circuits. *The Journal of Supercomputing* 73, 5 (2016), 1733–1759. doi:10.1007/s11227-016-1878-5
[37] Md Mahmud Muntakim Khan, Ayan Kumar Biswas, Shuvro Chowdhury, Mehbuba Tanzid, Kazi Mohammad Mohsin, Masud Hasan, and Asif Islam Khan. 2008. Quantum realization of some quaternary circuits. In *TENCON 2008-2008 IEEE Region 10 Conference*. IEEE, 1–5.
[38] Rashida Khanoma, Tahseen Kamalb, and Mozammel HA Khana. 2008. Genetic algorithm based synthesis of ternary reversible/quantum circuit. In *2008 11th International Conference on Computer and Information Technology*. IEEE, 270–275.
[39] M. V. Klymenko and F. Remacle. 2014. Quantum dot ternary-valued full-adder: Logic synthesis by a multiobjective design optimization based on a genetic algorithm. *Journal of Applied Physics* 116, 16 (Oct. 2014). doi:10.1063/1.4900995
[40] Emanuel Knill, Raymond Laflamme, and Gerald J Milburn. 2001. A scheme for efficient quantum computation with linear optics. *nature* 409, 6816 (2001), 46–52.
[41] Michael Kölle, Tom Bintener, Maximilian Zorn, Gerhard Stenzel, Leo Sünkel, Thomas Gabor, and Claudia Linnhoff-Popien. 2025. Evaluating Mutation Techniques in Genetic-Algorithm-Based Quantum Circuit Synthesis. In *Proceedings of the Genetic and Evolutionary Computation Conference* (NH Malaga Hotel, Malaga, Spain) *(GECCO '25)*. Association for Computing Machinery, New York, NY, USA, 907–915. doi:10.1145/3712256.3726402
[42] Rolf Landauer. 1961. Irreversibility and heat generation in the computing process. *IBM journal of research and development* 5, 3 (1961), 183–191.
[43] Soonchil Lee, Seong-Joo Lee, Taegon Kim, Jae-Seung Lee, et al. 2006. The Cost of Quantum Gate Primitives. *Journal of Multiple-Valued Logic & Soft Computing* 12 (2006).
[44] Jianing Li, Sijia Xu, Jiaming Zheng, Guoqing Jiang, and Weichao Ding. 2024. Research on Multi-Objective Evolutionary Algorithms Based on Large-Scale Decision Variable Analysis. *Applied Sciences* 14, 22 (2024), 10309.
[45] Martin Lukac, Michitaka Kameyama, D Michael Miller, and Marek A Perkowski. 2012. High Speed Genetic Algorithms in Quantum Logic Synthesis: Low Level Parallelization vs. Representation? *J. Multiple Valued Log. Soft Comput.* 20, 1-2 (2012), 89–120.
[46] Martin Lukac, Marek Perkowski, Hilton Goi, Mikhail Pivtoraiko, Chung Hyo Yu, Kyusik Chung, Hyunkoo Jee, Byung-Guk Kim, and Yong-Duk Kim. 2004. *Evolutionary Approach to Quantum and Reversible Circuits Synthesis*. Springer Netherlands, Dordrecht, 201–257. doi:10.1007/978-1-4020-2075-9_7
[47] Martin Lukac, Marek Perkowski, Hilton Goi, Mikhail Pivtoraiko, Chung Hyo Yu, Kyusik Chung, Hyunkoo Jeech, Byung-Guk Kim, and Yong-Duk Kim. 2003. Evolutionary approach to quantum and reversible circuits synthesis. *Artificial Intelligence Review* 20, 3 (2003), 361–417.

[48] Martin Lukac, Marek Perkowski, and Michitaka Kameyama. 2010. Evolutionary quantum logic synthesis of boolean reversible logic circuits embedded in ternary quantum space using structural restrictions. In *IEEE Congress on Evolutionary Computation*. IEEE, 1–8.
[49] Duncan L. MacFarlane, Hiva Shahoei, Ifeanyi G. Achu, Evan Stewart, Wiliam V. Oxford, and Mitchell A. Thornton. 2023. Multiple-Valued Logic Physically Unclonable Function in Photonic Integrated Circuits. In *2023 IEEE 53rd International Symposium on Multiple-Valued Logic (ISMVL)*. 184–189. doi:10.1109/ISMVL57333.2023.00043
[50] Sudhindu Bikash Mandal, Amlan Chakrabarti, and Susmita Sur-Kolay. 2012. A Synthesis Method for Quaternary Quantum Logic Circuits. In *Progress in VLSI Design and Test*, Hafizur Rahaman, Sanatan Chattopadhyay, and Santanu Chattopadhyay (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 270–280.
[51] Dmitri Maslov and Gerhard W Dueck. 2003. Garbage in reversible design of multiple output functions. In *6th International Symposium on Representations and Methodology of Future Computing Technologies*. 162–170.
[52] Dmitri Maslov and D. Michael Miller. 2006. Comparison of the Cost Metrics for Reversible and Quantum Logic Synthesis. arXiv:quant-ph/0511008 [quant-ph] https://arxiv.org/abs/quant-ph/0511008
[53] K. Matsui. 1999. New selection method to improve the population diversity in genetic algorithms. In *IEEE SMC'99 Conference Proceedings. 1999 IEEE International Conference on Systems, Man, and Cybernetics (Cat. No.99CH37028)*, Vol. 1. 625–630 vol.1. doi:10.1109/ICSMC.1999.814164
[54] D Michael Miller and Mitchell A Thornton. 2007. *Multiple-valued logic: Concepts and representations*. Morgan & Claypool Publishers.
[55] Michael D. Miller and Mitchell A. Thornton. 2008. *Multiple Valued Logic: Concepts and Representations*. Springer International Publishing. doi:10.1007/978-3-031-79779-8
[56] Majid Mohammadi and Mohammad Eshghi. 2008. Heuristic methods to use don't cares in automated design of reversible and quantum logic circuits. *Quantum Information Processing* 7, 4 (2008), 175–192.
[57] Majid Mohammadi and Mohammad Eshghi. 2009. On figures of merit in reversible and quantum logic designs. *Quantum Information Processing* 8, 4 (2009), 297–318.
[58] Asma Taheri Monfared, Valentina Ciriani, and Majid Haghparast. 2025. Balanced ternary reversible comparator for qutrit quantum circuits. *Journal of Physics A: Mathematical and Theoretical* 58, 24 (2025), 245305.
[59] Asma Taheri Monfared, Valentina Ciriani, Tommi Mikkonen, and Majid Haghparast. 2023. Quaternary reversible circuit optimization for scalable multiplexer and Demultiplexer. *IEEE Access* 11 (2023), 46592–46603.
[60] Debarati Mukherjee, Amlan Chakrabarti, and Debotosh Bhattacherjee. 2009. Synthesis of quantum circuits using genetic algorithm. *International Journal of Recent Trends in Engineering* 2, 1 (2009), 212.
[61] Ashok Muthukrishnan and Carlos R Stroud Jr. 2000. Multivalued logic gates for quantum computation. *Physical review A* 62, 5 (2000), 052309.
[62] Jitendra Nath and Tanay Chattopadhyay. 2013. *All-Optical Quaternary Logic Based Information Processing: Challenges and Opportunities*. InTech. doi:10.5772/51559
[63] Nayan Kumar Naware, Deepti S. Khurge, and S.U. Bhandari. 2015. Review of Quaternary Algebra & Its Logic Circuits. In *2015 International Conference on Computing Communication Control and Automation*. 969–973. doi:10.1109/ICCUBEA.2015.204
[64] Michael A. Nielsen and Isaac L. Chuang. 2000. *Quantum Computation and Quantum Information*. Cambridge University Press.
[65] Michael A Nielsen and Isaac L Chuang. 2010. *Quantum computation and quantum information*. Cambridge university press.
[66] Abdollah Norouzi Doshanlou, Majid Haghparast, Mehdi Hosseinzadeh, and Midia Reshadi. 2020. Efficient design of quaternary quantum comparator with only a single ancillary input. *IET Circuits, Devices & Systems* 14, 1 (2020), 80–87.
[67] Joseph L. Pachuau, Arnab Roy, and Anish Kumar Saha. 2021. An Overview of Crossover Techniques in Genetic Algorithm. In *Modeling, Simulation and Optimization*, Biplab Das, Ripon Patgiri, Sivaji Bandyopadhyay, and Valentina Emilia Balas (Eds.). Springer Singapore, Singapore, 581–598.
[68] Anouk Paradis, Benjamin Bichsel, Samuel Steffen, and Martin Vechev. 2021. Unqomp: synthesizing uncomputation in Quantum circuits. In *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation* (Virtual, Canada) *(PLDI 2021)*. Association for Computing Machinery, New York, NY, USA, 222–236. doi:10.1145/3453483.3454040
[69] Marek Perkowski et al. 2007. Realization of incompletely specified functions in minimized reversible cascades. In *Proc. Reed-Muller Workshop*. 59–65.
[70] Marek Perkowski, Lech Jozwiak, Pawel Kerntopf, Alan Mishchenko, Anas Al-Rabadi, Alan Coppola, Andrzej Buller, Xiaoyu Song, Svetlana Yanushkevich, Vlad P Shmerko, et al. 2001. A general decomposition for reversible logic. (2001).
[71] Marek Perkowski and Pawel Kerntopf. 2001. Reversible logic. Invited tutorial. *Proc. Euro-Micro* (2001).
[72] Stefania Perri, Pasquale Corsonello, and Giuseppe Cocorullo. 2014. Design of Efficient Binary Comparators in Quantum-Dot Cellular Automata. *IEEE Transactions on Nanotechnology* 13, 2 (2014), 192–202. doi:10.1109/TNANO.2013.2295711
[73] Fabio Pievani, Asma Taheri Monfared, Andrea Bombarda, and Angelo Gargantini. 2025. QuTiP-MRL: A Library for Multiple-Valued Reversible Logic Simulations. In *Euromicro Conference on Software Engineering and Advanced Applications*. Springer, 419–427.
[74] JH Plantenberg, PC De Groot, CJPM Harmans, and JE Mooij. 2007. Demonstration of controlled-NOT quantum gates on a pair of superconducting quantum bits. *Nature* 447, 7146 (2007), 836–839.

[75] I. Pogorelov, T. Feldker, Ch. D. Marciniak, L. Postler, G. Jacob, O. Krieglsteiner, V. Podlesnic, M. Meth, V. Negnevitsky, M. Stadler, B. Höfer, C. Wächter, K. Lakhmanskiy, R. Blatt, P. Schindler, and T. Monz. 2021. Compact Ion-Trap Quantum Computing Demonstrator. *PRX Quantum* 2, 2 (June 2021). doi:10.1103/prxquantum.2.020343

[76] Robin C. Purshouse and Peter J. Fleming. 2002. Why use elitism and sharing in a multi-objective genetic algorithm?. In *Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation* (New York City, New York) *(GECCO'02)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 520–527.

[77] H.G Rangaraju, Vinayak Hegde, K B Raja, and K N Muralidhara. 2012. Design of Efficient Reversible Binary Comparator. *Procedia Engineering* 30 (2012), 897–904. doi:10.1016/j.proeng.2012.01.943 International Conference on Communication Technology and System Design 2011.

[78] Cristian Ruican, Mihai Udrescu, Lucian Prodan, and Mircea Vladutiu. 2009. Genetic algorithm based quantum circuit synthesis with adaptive parameters control. In *2009 IEEE Congress on Evolutionary Computation*. IEEE, 896–903.

[79] P. Runeson and M. Höst. 2009. Guidelines for conducting and reporting case study research in software engineering. *Empirical Softw. Engg.* 14, 2 (April 2009), 131–164. doi:10.1007/s10664-008-9102-8

[80] Ankur Sarker, M. Shamiul Amin, Avishek Bose, and Nafisah Islam. 2014. An optimized design of binary comparator circuit in quantum computing. In *2014 International Conference on Informatics, Electronics & Vision (ICIEV)*. 1–5. doi:10.1109/ICIEV.2014.6850768

[81] Moein Sarvaghad-Moghaddam, Philipp Niemann, and Rolf Drechsler. 2018. Multi-objective synthesis of quantum circuits using genetic programming. In *International Conference on Reversible Computation*. Springer, 220–227.

[82] Trailokya Nath Sasamal, Ashutosh Kumar Singh, and Anand Mohan. 2015. Reversible Logic Circuit Synthesis and Optimization Using Adaptive Genetic Algorithm. *Procedia Computer Science* 70 (2015), 407–413. doi:10.1016/j.procs.2015.10.054 Proceedings of the 4th International Conference on Eco-friendly Computing and Communication Systems.

[83] Anupriya Shukla, Hari Mohan Pandey, and Deepti Mehrotra. 2015. Comparative review of selection techniques in genetic algorithm. In *2015 International Conference on Futuristic Trends on Computational Analysis and Knowledge Management (ABLAZE)*. 515–519. doi:10.1109/ABLAZE.2015.7154916

[84] Paolo Solinas, Paolo Zanardi, Nino Zanghì, and Fausto Rossi. 2003. Holonomic quantum gates: A semiconductor-based implementation. *Physical Review A* 67, 6 (June 2003). doi:10.1103/physreva.67.062315

[85] Asma Taheri Monfared, Andrea Bombarda, Angelo Gargantini, and Majid Haghparast. 2025. Efficient and scalable designs for ternary quantum reversible multiplexer and demultiplexer systems. *Quantum Information Processing* 24, 10 (Sept. 2025). doi:10.1007/s11128-025-04927-y

[86] Asma Taheri Monfared, Valentina Ciriani, Majid Haghparast, et al. 2024. Qutrit representation of quantum images: new quantum ternary circuit design. *Quantum Information Processing* 23, 8 (2024), 1–16.

[87] Asma Taheri Monfared, Majid Haghparast, and Kamalika Datta. 2019. Quaternary quantum/reversible half-adder, full-adder, parallel adder and parallel adder/subtractor circuits. *International Journal of Theoretical Physics* 58 (2019), 2184–2199.

[88] Himanshu Thapliyal, Edgard Muñoz-Coreas, and Vladislav Khalus. 2021. Quantum circuit designs of carry lookahead adder optimized for T-count T-depth and qubits. *Sustainable Computing: Informatics and Systems* 29 (2021), 100457.

[89] Himanshu Thapliyal and MB Srinivas. 2005. A novel reversible TSG gate and its application for designing reversible carry look-ahead and other adder architectures. In *Asia-Pacific conference on advances in computer systems architecture*. Springer, 805–817.

[90] D. Thierens. 2002. Adaptive mutation rate control schemes in genetic algorithms. In *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No.02TH8600)*, Vol. 1. 980–985 vol.1. doi:10.1109/CEC.2002.1007058

[91] Hao Wang, Chaoli Sun, Haibo Yu, and Xiaobo Li. 2022. A decomposition-based many-objective evolutionary algorithm with optional performance indicators. *Complex & Intelligent Systems* 8, 6 (2022), 5157–5176.

[92] Andrew G White, Alexei Gilchrist, Geoffrey J Pryde, Jeremy L O'Brien, Michael J Bremner, and Nathan K Langford. 2007. Measuring two-qubit gates. *Journal of the Optical Society of America B* 24, 2 (2007), 172–183.

[93] Robert Wille and Rolf Drechsler. 2009. BDD-based synthesis of reversible logic for large functions. In *Proceedings of the 46th Annual Design Automation Conference* (San Francisco, California) *(DAC '09)*. Association for Computing Machinery, New York, NY, USA, 270–275. doi:10.1145/1629911.1629984

[94] Robert Wille, Eleonora Schönborn, Mathias Soeken, and Rolf Drechsler. 2016. SyReC: A hardware description language for the specification and synthesis of reversible circuits. *Integration* 53 (2016), 39–53.

[95] Christian Wood and Alberto Moraglio. 2025. Noise resilient quantum circuit design by multi-objective genetic algorithm. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion* (NH Malaga Hotel Malaga Spain). ACM, New York, NY, USA, 2612–2615.

[96] Ge Yan, Wenjie Wu, Yuheng Chen, Kaisen Pan, Xudong Lu, Zixiang Zhou, Yuhan Wang, Ruocheng Wang, and Junchi Yan. 2025. Quantum Circuit Synthesis and Compilation Optimization: Overview and Prospects. arXiv:2407.00736 [quant-ph] https://arxiv.org/abs/2407.00736

[97] Christof Zalka. 1999. Grover's quantum searching algorithm is optimal. *Physical Review A* 60, 4 (1999), 2746.

[98] Ricardo Salem Zebulum, Marco Aurélio Pacheco, and Marley Maria Be Vellasco. 2018. *Evolutionary electronics: automatic design of electronic circuits and systems by genetic algorithms*. CRC press.

[99] Tingyan Zhang, Yi Li, Sijun Lu, Jingwen Ai, Mingqiang Bai, Zhiwen Mo, Wenbo Du, Pengheng Jiang, and Jiawei Liu. 2025. Optimization and design for multi-valued quantum comparator circuits. *Journal of Physics A: Mathematical and Theoretical* 58, 4 (2025), 045303.
[100] Shi-Liang Zhu and Z. D. Wang. 2002. Implementation of Universal Quantum Gates Based on Nonadiabatic Geometric Phases. *Physical Review Letters* 89, 9 (Aug. 2002). doi:10.1103/physrevlett.89.097902

Received 17 November 2025; revised 1 May 2026; accepted 2 May 2026