IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received November 28, 2019, accepted December 21, 2019, date of publication December 25, 2019,
date of current version January 3, 2020.

Digital Object Identifier 10.1109/ACCESS.2019.2962155

A Review of Quantum-Inspired Metaheuristics: Going From Classical Computers to Real Quantum Computers

OSCAR H. MONTIEL ROSS, (Senior Member, IEEE)
Centro de Investigación y Desarrollo de Tecnología Digital—Instituto Politécnico Nacional (CITEDI-IPN), Tijuana 22435, México
e-mail: oross@ipn.mx

This work was supported in part by the Instituto Politécnico Nacional, and in part by the Consejo Nacional de Ciencia y Tecnología (CONACYT).

ABSTRACT This paper presents a review of quantum-inspired population-based metaheuristics. Quantum-inspired algorithms were born when there were no quantum computers; they demonstrated to have interesting characteristics providing good results in classical computers. At present, when the first quantum computers are available, scientists are working to confirm the quantum supremacy in different fields. After almost 20 years that the first metaheuristic inspired in quantum phenomena was published, a large number of works have been proposed. This paper aims to look back to see which quantum-inspired metaheuristics could be translated to be used in the existing quantum computers based on the circuit model programming paradigm. Reviewed metaheuristics were classified according to their main source of inspiration; just some representative works of each classification were selected because of the vast number of existing works on each one. The analysis was done for the circuit model and metrics as width, size, and length were used to determine their viability of being implemented in a real quantum computer. Moreover, comparative results using metrics such as performance and running time for quantum-inspired metaheuristic were included.

INDEX TERMS Quantum metaheuristic, quantum-inspired, circuit model, metaheuristic for quantum computer, binary coded quantum-inspired metaheuristic, real coded quantum-inspired metaheuristic.

I. INTRODUCTION

Mathematical optimization plays a crucial role in solving science, engineering, economics, and life problems since they can be formulated as a search or optimization problems. Traditional optimization methods can solve some simple problems using exact methods. Most problems are complex, and exact methods fail to solve them because they have properties such as multimodality, high-dimensionality, and non-differentiability.

An approximation algorithm is a procedure that can provide an approximate solution to a given problem. Rather than spending an exponential amount of time searching the optimal solution, approximation algorithms run in polynomial time according to the input size and settle solutions near the optimal [70]; they can be divided into heuristics and metaheuristics algorithms. Heuristics stands for the art of discovering new strategies for solving problems based on experience. They can give a satisfactory solution that might not be optimal in an acceptable quantity of time; in this category, we can found many exact and approximation algorithms. The term metaheuristic is formed by the words ''meta'' meaning upper level, and ''heuristic''. Therefore, metaheuristic refers to a collection of methodologies conceptually positioned above heuristics [16]. Metaheuristics can be of a single solution or population-based. In the first case, they are based on a single solution proposal and regarded to be more exploitative and embrace methods such as simulated annealing; in contrast, metaheuristics based on populations comprise methods such as evolutionary algorithms (EAs) and swarm-based algorithms; these metaheuristics are more oriented to the exploitation of the search space.

The importance of metaheuristic algorithms was fully understood until the NP-completeness theory was established in 1971. In this theory was determined that many well-known problems are intractable in the sense that in polynomial time,

The associate editor coordinating the review of this manuscript and approving it for publication was Kun Wang.

no optimal solution can be computed for them [10]. As a consequence, approximation algorithms that provide near-optimal solutions are the best option when trying to solve these problems. In the past decades, the area of approximation algorithms experienced an explosive rate of growth. Several factors, such as the development of the field of data mining, bioinformatics, deep learning, and others, triggered this growth. As a result, a significant number of intractable optimization problems surged, most of which have direct application to real-world problems.

Overall, to evaluate and compare algorithms, it is necessary to use objective metrics that measure how well the algorithms are behaving and tell us what to expect for the general case. For exact (non-approximate) algorithms, an important metric when finding exact solutions is the **running time**. The evaluation of approximate algorithms requires a second metric that measures the **performance** of the algorithms. That is, it is essential to measure how close the output of the approximation algorithm is from the optimal solution. Therefore, it is common to test the algorithmic proposals using sets of benchmark functions. These functions face the algorithms to different challenges, and the metrics help to determine the strengths and weaknesses of the approximation algorithms.

Nature has inspired many approximation algorithm metaheuristics that can solve successfully hard problems. Natural computing is concerned with computational processes inspired by nature. Some well-known examples of this discipline are: neural networks, swarm intelligence, artificial immune system, evolutionary computation, cellular automata, molecular, membrane, and quantum computing. These, together with other paradigms in the same line constitute the discipline of computational intelligence.

As we mentioned, execution time is an essential metric when designing exact and approximation algorithms. Undoubtedly, advances in computer architecture have played a significant role that has impacted this metric positively. Algorithms designed for sequential processing have been modified to take advantage of parallel processing. These new modified algorithms use multicore microprocessor architectures, and more recently, computer cards containing thousands of processing units known as GPU cards and specialized software such as the CUDA [17]. The development of quantum computers by companies like the IBM [30], D-Wave Systems [11], Google [21], QuTech [55], Intel [31], and Rigetti [59] is becoming a watershed in some fields like optimization, machine learning, material sciences, and Monte Carlo simulations. The interest in developing quantum applications is justified since there is experimental evidence that shows that one D-Wave quantum computer system outperformed software as the CPLEX of IBM when solving three instances in the NP-hard problem domain. In one of the tests, the quantum computer solved 3600 times faster than the CPLEX software [43].

Quantum metaheuristics are approximate algorithms designed to impact both metrics, running time, and performance, since they will be executed by a quantum computer. However, quantum-inspired metaheuristics were designed to run in classical computers that simulating phenomena of quantum physics like superposition and entanglement allow us to explore quantum programming and predict future behavior and results.

This paper aims to present a review of state-of-the-are about quantum-nature-inspire metaheuristics. The number of existing proposals is very extensive to include all of them. We made a selection based on criteria such as: including pioneer works, representative works of every classification, state-of-the-art works, and other interesting proposals that reported good results. We tried to increase the number of the included works; unfortunately, we had to leave out many interesting works that have contributions so as not to extend so much this paper. The review is divided into five groups, according to their source of inspiration, all of them are quantum-inspired. We provide a summary and analysis of the reported results of the chosen metaheuristics. We comment on which ones could be susceptible to be adapted for being executed by a quantum computer that uses the circuit model.

There are some surveys about quantum-inspired metaheuristics that contains useful information about the performance [35], [75]. We did not find a survey that reported the comparison of running times. In this paper, the reader could discover some newer works, which were not included in the surveys because they were published after; we present information about the performance and running times. The main contribution is the analysis regarding looking back and see which of the existing metaheuristics could be translated for being used in today's quantum computers. This study is limited to those quantum computers based on the circuit model, and objective metrics are used to make the discussion.

In Section II, we provide the minimal theoretical background of quantum computing to make more understandable quantum algorithms. In Section III, we present the pioneering work of all the quantum-inspired population-based metaheuristics. This section also includes other binary and real coded evolutionary algorithms. Section IV is dedicated to quantum-inspired swarm evolutionary algorithms. In Section V the quantum social evolutionary algorithm is explained. In Section VI, we explain the quantum-inspired multiobjective evolutionary algorithm. In Section VII, the algorithms inspired by other sources of inspiration such as the gravity force. In Section VIII, the analysis of the revised method is achieved. Finally, Section IX contains the conclusions of this work.

II. THEORETICAL BACKGROUND

Any evolutionary algorithm, in its more basic representation can be modeled for computer simulation using the difference equation shown in (1) [18], where t is the generation counter, $P(t+1)$ is the new population obtained from the current population $P(t)$ after being modified by random variations v and a selection process s. This equation can also represent other nature-inspired population-based algorithms, which, rather than using the evolution mechanism, are based on

different mechanisms such as swarm behavior; it is also valid for quantum metaheuristic algorithms inspired on physical phenomena

$$P(t+1) = s(v(P(t))). \tag{1}$$

Real-life applications require to use classical information based on bits. Data from a quantum register memory cannot be directly used because it can have any possible value. An observer can obtain classical information from a quantum register. The observer must be equipped with the appropriated measurement apparatus to determine the outcome of variables representing observable physical quantities such as energy, momentum, position, and others. These variables are replaced by linear operators named *observables* in the Hilbert space. The system has associated an element ψ of the Hilbert space, and it describes the quantum state of the system. ψ is known as the system's state vector or wavevector or wavefunction. The only values that can be observed from ψ for physical quantities are the eigenvalues of the corresponding operators on the Hilbert space. In agreement with the Copenhagen interpretation of Quantum Mechanics, an abstract wavefunction allows calculating probabilities outcomes of particular experiments. $|\psi|^2$ is a probability density function (PDF). As an example, we will consider a particle; the PDF will describe the probability than an observation of such a particle will be found at a given time in a region of the space. ψ satisfies the Schrödinger linear equation

$$i\hbar\frac{\partial}{\partial t}\psi = \mathcal{H}\psi; \tag{2}$$

where $i = \sqrt{-1}$, $\hbar$ is the reduced Planck's constant, and $\mathcal{H}$ is the Hamiltonian. Equation (2) describes the time evolution of the wavefunction along with the PDF. At each point, as time passes, the PDF becomes more disperse over space, and our capacity to determine the position of the particle becomes less precise. This dispersion or spreading will continue until an observation is achieved. Then, ψ collapses to a particular eigenstate (classical state), in the case of the particle, in a particular position, then, the dispersing of the PDF starts all over again.

Before a quantum system is observed, it is a linear combination of all possible states. When we performed an observation (measurement), one classical state is chosen, with a probability given by the PDF. The likelihood of finding the observed particle is more significant at locations where $|\psi|^2$ is large, and the probability is zero if $|\psi|^2 = 0$. Finding a particle in any place in space has a probability value of 1; therefore, the normalized wavefunction ψ is used. All the operators in the Hilbert space are unitary.

Quantum computing uses the Dirac notation; hence, a qubit $|\psi\rangle$ is a two-state system defined as shown in (3); the complex coefficient a is the amplitude of the $|0\rangle$ component, and b is the amplitude of the $|1\rangle$. It is required that qubits be normalized, so they must fulfill that $|a|^2 + |b|^2 = 1$ to ensure properly this condition.

Quantum algorithms work with quantum bits (qubits). A qubit is an elementary information unit, and it is analog to a bit in classical computation. Qubits can be in a superposition of the orthogonal states $|0\rangle$ and $|1\rangle$ simultaneously; hence, a qubit $|\psi\rangle$ may be defined as the complex $\mathbb{C}$ combination of both states in the Hilbert space,

$$|\psi\rangle = a|0\rangle + b|1\rangle \quad \text{where} \quad a, b \in \mathbb{C}. \tag{3}$$

The state of a qubit can be visualized inside of a bounding sphere, known as the Bloch sphere. Therefore, a qubit can be expressed in the spherical coordinate system (r, θ, ϕ). r is the radial distance; θ is the latitude measured monotonically from North-pole to South-pole, $0 \geq \theta \leq \phi$; and π is the longitude measured monotonically as we rotate around the z-axis in a clockwise fashion. Because the normalization condition, we have that $x^2 + y^2 + z^2 = r^2 = 1$, hence $r = 1$. To switch from spherical coordinates to Cartesian coordinates, we have that,

$$x = r\sin(\theta)\cos(\theta) \tag{4}$$
$$y = r\sin(\theta)\sin(\phi)) \tag{5}$$
$$z = r\cos(\theta). \tag{6}$$

In the Cartesian coordinate system, in the z-axis, a qubit is in the north-pole when its state is $|1\rangle$ when $z = 1$, and in the south-pole for the state $|0\rangle$ when $z = -1$.

Any pair of vectors ϕ and ψ that are linearly independent $\in \mathbb{C}^2$ can form a basis. It is very common to use the orthonormal basis $|0\rangle$ and $|1\rangle$ known as the computational basis. Some other bases are the Bell, Chiral, and Diagonal basis. To extract answers from quantum computers is necessary to read the quantum computing register; so, we must measure the quantum state of the register, and the result must be expressed on a specific basis. Therefore, the bases are essential to the definition of the measurement operator. The measurement will destroy the pure quantum state collapsing to non-quantum values on the selected basis.

Classical computing works with a set of bits to form registers, some of them known as nibbles, bytes, and words. A three-bit system of classical bits, say 000 or 101, always has the same value unless we intentionally modify them, and to represent all the possible combinations, we require a system with eight registers. In quantum computing, a set of qubits forms a quantum register, which can have all the possible combinations that their binary counterpart could have with the same number of bits. Hence, one quantum memory register with 16 qubits will hold the same information that 2^{16} binary memory registers.

In general, a pure state of an n-qubit quantum memory register in the computational basis is written as

$$|\Psi\rangle = c_0|00\ldots0\rangle + c_1|00\ldots1\rangle + \cdots + c_{2^n-1}|11\ldots1\rangle = \sum_{i=0}^{2^n-1} c_i|i\rangle; \tag{7}$$

where $\sum_{i=0}^{2^n-1} |c_i|^2 = 1$ to satisfy the normalization condition, and $|i\rangle$ is the computational basis eigenstate whose bits correspond those of the decimal number i in base-2 notation; for example, for a three qubit system $|2\rangle$ corresponds to $|010\rangle$.

The state of a quantum register is determined by calculating the direct product symbolized by $\otimes$, which is also known as tensor or Kroenecker product of the n quantum states of the individual qubits. If $|\phi\rangle = \sum_{j=0}^{2^m-1} a_j|j\rangle$ is an m-qubit register and $|\psi\rangle = \sum_{k=0}^{2^n-1} b_k|k\rangle$ is an n-qubit register, both in pure state, their direct product $|\phi\rangle \otimes |\psi\rangle$ is calculated as shown in (8)

$$|\phi\rangle \otimes |\psi\rangle = \sum_{j=0}^{2^m-1} a_j|j\rangle \otimes \sum_{k=0}^{2^n-1} b_k|k\rangle = \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_{2^m-1} \end{pmatrix} \otimes \begin{pmatrix} b_0 \\ b_1 \\ \vdots \\ b_{2^m-1} \end{pmatrix}. \quad (8)$$

Most of the quantum-inspired algorithms follow the circuit model of a computer as an abstraction of the computing process. In the most general form, in this model, the computations are achieved by the application of different gates (binary or quantum) to transform the inputs in some fashion.

There are different gates that can be built from unitary matrices to act on qubits. In this paper, we will focus mainly on those unitary matrices and quantum gates that will be used in the explanation of the included algorithms. Therefore, we have included the Pauli matrices: Identity $\mathbb{I} = \left(\begin{smallmatrix} 1 & 0 \\ 0 & 1 \end{smallmatrix}\right)$, single qubit rotation around x-axis π radians $X = \left(\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}\right)$, rotation around y-axis π radians $Y = \left(\begin{smallmatrix} 0 & -i \\ i & 0 \end{smallmatrix}\right)$, and rotation around z-axis π radians $\mathbb{Z} = \left(\begin{smallmatrix} 1 & 0 \\ 0 & -1 \end{smallmatrix}\right)$. The Hadamard gate ($H$) that acts on the computational basis according to $H|x\rangle = \frac{1}{\sqrt{2}}(|0\rangle + (-1)^x|1\rangle)$. Rotation gates: $R_x(\theta)$ that allows rotation of θ radians around the x-axis, $R_y(\theta)$ that allows rotation of θ radians around y-axis, and $R_z(\theta)$ that allows rotation of θ radians around z-axis; their definition is as follows,

$$R_x(\theta) = \begin{pmatrix} \cos(\theta/2) & -i\sin(\theta/2) \\ -i\sin(\theta/2) & \cos(\theta/2) \end{pmatrix} \quad (9)$$

$$R_y(\theta) = \begin{pmatrix} \cos(\theta/2) & \sin(\theta/2) \\ \sin(\theta) & \cos(\theta/2) \end{pmatrix} \quad (10)$$

$$R_z(\theta) = \begin{pmatrix} e^{-i\frac{\theta}{2}} & 0 \\ 0 & e^{i\frac{\theta}{2}} \end{pmatrix}. \quad (11)$$

The Hadamard gate is very important. We can prepare n-qubits in the state $|0\rangle$ and apply to each qubit in parallel its own H gate as shown in (12) to produce an equal superposition state of all the qubits in the register

$$H|0\rangle \otimes H|0\rangle \otimes \cdots \otimes H|0\rangle = H^{\otimes n}|0\rangle = \frac{1}{\sqrt{2^n}} \sum_{j=0}^{2^n-1} |j\rangle. \quad (12)$$

In (12), the number j is in base-10 notation, and $|j\rangle$ is the computational basis state, in such a way that in an 8-qubit register the state $|18\rangle$ corresponds to the computational basis state $|00010010\rangle$. It is important to prepare n − *qubits* registers in the state $|0\rangle$, and then apply to each one the H gate. With this operation, we create one register containing 2^n component eigenstates that represents all the possible combinations of bit-strings one can store in 2^n different memory registers [69].

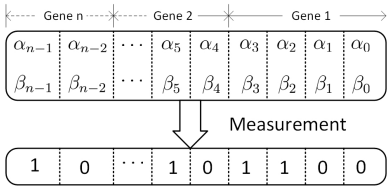

FIGURE 1. Quantum chromosome structure. Each gene represents a decision variable.

All the metaheuristic algorithms explained in this work used the computational basis; therefore, we limit the explanation of observing the states of qubits or quantum register to this basis. In the computational basis, the measurements of a qubit are achieved in the z-axis, corresponding to $|0\rangle$ or $|1\rangle$, as more significant the value of $|a|^2$, the probability of being "0" in the classical state increases. If the measurement is applied to an n-qubit quantum memory register, all the possible 2^n bit string configurations can be obtained.

Fig. 1 describes a quantum chromosome conveniently defined as a quantum register containing n qubits. Therefore, it is possible to modify standard metaheuristics to work with few or even a single chromosome (quantum register) instead of having a large population of solution encoded as is done in non-quantum evolutionary algorithms. The capacity of a quantum register to represent multiple states simultaneously helps to maintain diversity during the search process of the evolutionary algorithm.

III. QUANTUM-INSPIRED EVOLUTIONARY ALGORITHM

Narayanan and Moore in 1996 introduced the concept and principles of quantum mechanics to obtain more efficient evolutionary methods [51]. Specifically, they proposed a quantum-inspired form of fixed-point interference crossover operator that was applied to solve the traveling salesman problem (TSP). In [50], the first guidelines as an attempt to characterize a methodology to design and develop quantum algorithms were provided, and it was explained the feasibility and potential use of quantum-inspired methods in tackling NP-hard problems.

The first two pioneer works based on concepts and principles of quantum computing, such as quantum bits and the superposition of states that include pseudocode are *genetic quantum algorithm* (GQA) proposed in [22], and *quantum inspired evolutionary algorithm* (QEA) introduced in [23].

A. THE GENETIC QUANTUM ALGORITHM (GQA)

Algorithm 1 shows the canonical structure of the GQA [22]. Analogously to the classical binary GA, a quantum

chromosome $\mathbf{q}_j$ at generation t is a set of qubits $|\psi_i\rangle$; such that,

$$\mathbf{q}_j^t = \begin{bmatrix} \alpha_1^t & \alpha_2^t & \cdots & \alpha_m^t \\ \beta_1^t & \beta_2^t & \cdots & \beta_m^t \end{bmatrix}; \quad (13)$$

where j is the chromosome identifier and m is the number of qubits in $\mathbf{q}_j^t$. A quantum chromosome $\mathbf{q}_j^t|_{t=0}$ represents the linear superposition of all possible states with the same probability, such that the state of the quantum chromosome $|\Psi_{\mathbf{q}_j^0}\rangle$ is,

$$|\Psi_{\mathbf{q}_j^0}\rangle = \sum_{k=1}^{2^m} \frac{1}{\sqrt{2^m}} |X_k\rangle; \quad (14)$$

where X_k represents the k-th state of a binary string $(x_1 x_2 \cdots x_m)$, and the bit value of x_i, for $i = 1, 2, \ldots, m$, is either 0 or 1. A population of quantum chromosomes is denoted by $Q(t) = \{\mathbf{q}_1^t, \mathbf{q}_2^t, \ldots, \mathbf{q}_n^t\}$; n is the size of the population.

Algorithm 1 Genetic Quantum Algorithm (GQA)

```
1  ProcedureGQA
2      t ← 0
3      initialize the population of quantum chromosomes
       Q(t)
4      make P(t) by observing Q(t) states
5      evaluate each binary solution in P(t)
6      store the best solution b among P(t)
7      while not termination-condition do
8          t ← t + 1
9          make P(t) by observing Q(t − 1) states
10         evaluate each binary solution in P(t)
11         update Q(t) using quantum gates U(t)
12         store the best b solution among P(t)
13     end
14 QEnd_GQA
```

Algorithm 2 Procedure Make for the GQA

```
1  Proceduremake(x)
2      i ← 0
3      while i < m do
4          i ← i + 1
5          if random[0, 1) > |α_i|^2 then
6              x_i ← 1
7          else
8              x_i ← 0
9          end
10     end
11 End_make
```

Some specific details of Algorithm 1 are described next:

1) The generation counter t is initialized in 0.
2) In line 3, a population of quantum chromosomes $Q(t)$ is created such that $\mathbf{q}_j^0$ for all $j = 1, 2, \ldots, n$ are initialized with $1/\sqrt{2}$ using (14).
3) The initial population $P(0) = \{\mathbf{x}_1^0, \mathbf{x}_2^0, \cdots, \mathbf{x}_n^0\}$ is obtained by observing $Q(0)$ states. Each $\mathbf{x}_j^0$, $j = 1, 2, \cdots, n$, is a solution, i.e, a binary chromosome of length m. The observations of quantum chromosomes are performed using the procedure **make** shown in Algorithm 2, it uses the probabilities of each qubit of $\mathbf{q}_j^0$ to be 0 or 1, i.e., $|\alpha_i^0|^2$ or $|\beta_i^0|^2$, respectively.
4) Each binary solution $\mathbf{x}_j^0$ in $P(0)$ is evaluated to obtain a quality measure through a fitness function.
5) The best binary solution in $P(0)$ is selected and stored into **b**.
6) The while loop contains the next steps.
   a) The generation counter t is incremented.

Algorithm 3 Procedure Update of the GQA

```
1  Procedureupdate(q)
2      i ← 0
3      while i < m do
4          i ← i + 1
5          determine θ_i with the lookup table
6          obtain (α'_i β'_i) as:
7          [α'_i β'_i]^T = U(θ_i)[α_i β_i]^T
8      end
9      q=q'
10 End_update
```

Algorithm 4 Repair Operator Used by the GQA, and the QEA

```
1  Procedurerepair(x)
2      knapsack-overfilled ← false
3      if Σ_{i=1}^m w_i x_i > C then
4          knapsack-overfilled ← true
5      end
6      while knapsack-overfilled do
7          select an i − th item from the knapsack
8          x_i ← 0
9          if Σ_{i=1}^m w_i x_i ≤ C then
10             knapsack-overfilled ← false
11         end
12     end
13     while not knapsack-overfilled do
14         select an i − th item from the knapsack
15         x_i ← 1
16         if Σ_{i=1}^m w_i x_i > C then
17             knapsack-overfilled ← true
18         end
19     end
20     x_j ← 0
21 End_repair
```

TABLE 1. Lookup table to find the appropriated rotation angle $\Delta\theta_i$.

x_i	b_i	$f(\mathbf{x}) \leq f(\mathbf{b})$	$\Delta\theta_i$	$s(\alpha_i\beta_i)$			
				$\alpha_i\beta_i \leq 0$	$\alpha_i\beta_i \leq 0$	$\alpha_i = 0$	$\beta_i = 0$
0	0	false	0	0	0	0	0
0	0	true	0	0	0	0	0
0	1	false	0	0	0	0	0
0	1	true	0.05π	-1	+1	±1	0
1	0	false	0.01π	-1	+1	±1	0
1	0	true	0.025π	+1	-1	0	±1
1	1	false	0.005π	+1	-1	0	±1
1	1	true	0.025π	+1	-1	0	±1

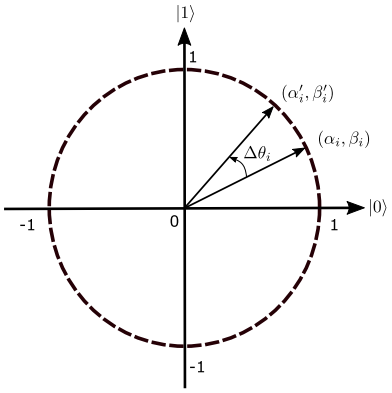

FIGURE 2. Polar plot that illustrates the angle $\Delta\theta_i$ when the rotation gate is applied to a qubit.

b) The procedure **make** creates a new population $P(t)$ by observing $Q(t-1)$ states.
c) Each binary solution of $P(t)$ is evaluated through a fitness function.
d) The aim of the **update** procedure is improving $Q(t)$ through generations. Its implementation will depend on the given problem. For example, Algorithm 3 was used in [22]; where the **update** procedure uses the actual $\mathbf{q}_j^t$, the binary solution $\mathbf{x}_j = \{x_1, \ldots, x_n\}$, an the best solution $\mathbf{b} = \{b_1, \ldots, b_n\}$ to calculate $f(\mathbf{x})$ as well as $f(\mathbf{b})$. The rotation angle θ_i and sign are determined with the lookup table shown in Table 1. The update of each qubit of $\mathbf{q}_j^t$ is carried out in line 7 of Algorithm 3 where the quantum gate $U(\xi\,\Delta\theta_i)$ is applied as shown in (15). Fig. 2 illustrates this concept. The angle $\Delta\theta_i$ of Table 1 is chosen in compliance with the application problem. Note that $\xi(\Delta\theta_i) = s(\alpha_i, \beta_i) * \Delta_i$, where the terms $s(\alpha_i, \beta_i)$ and $\Delta\theta_i$ are the direction and rotation angles, respectively.
e) The best solution among $P(t)$ is stored in $\mathbf{b}$.

$$\begin{bmatrix} \alpha_i' \\ \beta_i' \end{bmatrix} = \begin{bmatrix} \cos\xi(\Delta\theta_i) & -\sin\xi(\Delta\theta_i) \\ \sin\xi(\Delta\theta_i) & \cos\xi(\Delta\theta_i) \end{bmatrix} \begin{bmatrix} \alpha_i \\ \beta_i \end{bmatrix} \tag{15}$$

The GQA was adapted and tested to solve the 0-1 knapsack problem; which is defined as "given weights and values of n items and a knapsack, find a subset of the items that maximize the profit $f(\mathbf{x})$"; i.e.,

$$f(\mathbf{x}) = \sum_{i=1}^{m} p_i x_i$$

subject to

$$\sum_{i=1}^{m} w_i x_i < C$$

where the vector $\mathbf{x} = (x_i, \ldots, x_m)$, $x_i \in \{0, 1\}$, $\forall i \leq i < n$, w_i is the weight of item i, p_i is the profit of item i, and C is the knapsack capacity.

To adapt Algorithm 1 for solving the 0-1 kanpsack problem, it is necessary to add the repair operator described by the Algorithm 4 between lines (4 - 5) and (9-10). A first version of the repair operator was originally proposed in [44].

B. QUANTUM-INSPIRED EVOLUTIONARY ALGORITHM (QEA)

The QEA [23] is the upgrade of the GQA of [22]; similarly to its predecessor, the QEA was tested with the 0-1 knapsack combinatorial optimization problem. A notable difference between both proposals is the concept of *migration* introduced by the QEA, which is a process that can induce a variation of the probabilities of a quantum chromosome. The migration condition is a design parameter defined in the paper as follows:

"A migration in QEA is defined as the process of coping $\mathbf{b}_\mathbf{j}^\mathbf{t}$ in $B(t)$ or $\mathbf{b}$ to $B(t)$. A *global migration* is implemented by replacing all the solutions in $B(t)$ by $\mathbf{b}$. A *Local migration* is implemented by replacing some of the solutions in $B(t)$ by the best one of them [23]."

This work introduced the concept of "Q-bit individual" defined as a string of qubits (Q-bit); i.e., a Q-bit individual is a quantum chromosome defined by (13). The overall structure of the QEA is shown in Algorithm 5. This algorithm has similarities with its predecessor GQA (Algorithm 1). Hence the explaining of lines 2 to 5 of Algorithm 5 is the same of Algorithm 1.

In line 6 of Algorithm 5, the initial best solutions among $P(0)$ are saved into $B(0)$. This will form a list of binary solutions $B(0) = \{\mathbf{b}_1^0, \mathbf{b}_2^0, \ldots, \mathbf{b}_n^0\}$. At the initial generation $\mathbf{x}_j^0$ is the same as $\mathbf{b}_j^0$.

Algorithm 5 Quantum-Inspired Evolutionary Algorithm (QEA)

```
1  ProcedureQEA
2      t ← 0
3      initialize the population of quantum chromosomes Q(t)
4      make P(t) by observing the states of Q(t)
5      evaluate P(t)
6      store the best solutions among P(t) into B(t)
7      while not termination-condition do
8          t ← t + 1
9          make P(t) by observing the states of Q(t − 1)
10         evaluate the population of solutions P(t)
11         update Q(t) using quantum gates
12         store the best b solutions among B(t − 1) and P(t) into B(t)
13         store the best solution b among B(t)
14         if migration-condition then
15             migrate b or b_j^t to B(t) globally or locally respectively
16         end
17     end
18 AEnd_QEA
```

Algorithm 6 Procedure Update of the QEA

```
1  Procedureupdate(q)
2      i ← 0
3      while i < m do
4          i ← i + 1
5          determine Δθ_i with the lookup table
6          obtain (α'_i β'_i) from the following
7          if q is located in the first/third quadrant then
8              [α'_i β'_i]^T = U(Δθ_i)[α_i β_i]^T
9          else
10             [α'_i β'_i]^T = U(−Δθ_i)[α_i β_i]^T
11         end
12     end
13     q=q'
14 End_update
```

The **while** loop contains the next steps:

1) The generation counter t is updated.
2) A binary population $P(t)$ is created by observing the states of $Q(t-1)$ using the procedure **make** shown in Algorithm 2; in this version, the authors changed the original line 5 by the next pseudocode "**if** *random*$[0, 1] < |\beta|^2$ **then**". This expression produce almost the same result than the original proposal.
3) The binary population $P(t)$ is evaluated using a fitness function.
4) The procedure **update** shown in Algorithm 6, modifies the quantum chromosomes $Q(t)$ using quantum gates. In [23], the procedure **update** is an upgrade of the equivalent procedure shown in Algorithm 3 of the GQA. The new procedure **update** (Algorithm 6) reduces the size of the original lookup Table 1. The new lookup table to solve the 0-1 knapsack problem is shown in Table 2.
5) The best solution **b** among $B(t-1)$ and $P(t)$ are saved into $B(t)$; if the best solution saved in $B(t)$ is better than the stored best solution **b** then the saved solution **b** is substituted by the new one.
6) In the last step of the **while** loop, the algorithm verifies if there exists a migration condition. If the conditional is satisfied, the best solution **b** is migrated to $B(t)$, or the best one in $B(t)$ is migrated to them.

TABLE 2. Lookup table of $\Delta\theta_i$. x_i and b_i are the $i-th$ bits of the binary solution x and the best solution b, respectively. $f(\cdot)$ is the fitness function (profit). For the knapsack problem the next values were used: $\theta_1 = 0$, $\theta_2 = 0$, $\theta_3 = 0.01\pi$, $\theta_4 = 0$, $\theta_5 = -0.01\pi$, $\theta_6 = 0$, $\theta_7 = 0$, $\theta_8 = 0$.

x_i	b_i	$f(\mathbf{x}) \leq f(\mathbf{b})$	$\Delta\theta_i$
0	0	false	θ_1
0	0	true	θ_2
0	1	false	θ_3
0	1	true	θ_4
1	0	false	θ_5
1	0	true	θ_6
1	1	false	θ_7
1	1	true	θ_8

The adaptation of Algorithm 5 to solve the 0-1 knapsack problem besides the modification above mentioned; similarly to the GQA, it is also necessary to add the procedure **repair** (Algorithm 4) between lines (4-5) and (9-10).

The selection process of the vector of angles $\Theta = [\theta_1 \cdots \theta_8]^T$ is intuitive.

C. REAL-CODED QUANTUM-INSPIRED EVOLUTION ALGORITHM

The first successful work that used real-values in a quantum evolutionary algorithm was [1]; here, an algorithm with better convergence times when optimizing real-valued benchmark functions was presented. Based on the previous work, in [2] a simplified representation was used, and the proposal was named Real-coded quantum-inspired evolution algorithm (RQIEA); it was presented as an alternative to the binary-coded QIEA since the real-coded algorithm works better when real numbers are directly encoded into the chromosome. Additionally, the demand for memory is reduced.

Similarly to the binary-valued QIEA, the RQIEA maintains two different populations. In other words, the RQIEA maintains a distinction between the population formed by quantum chromosomes $Q(t)$ and the observed population that contains the real-valued solution vectors. In the RQIEA, $Q(t)$ has N quantum individuals $q_i, i = 1, 2, \ldots, N$. Each individual is composed by G genes g_{ij} where $j = 1, 2, \ldots, G$. Each gene is represented by a pair $g_{ij} = (\rho_{ij}, \sigma_{ij})$ of real numbers where the mean and width of a squared pulse are symbolized by ρ_{ij} and σ_{ij}, respectively. In the quantum individual, each gene represents an interval in the search space. The main

idea is to have similitude with quantum concepts like the superposition of states and collapsing to a value.

Algorithm 7 shows the pseudocode of the RQIEA. Similarly to other population-based optimization algorithms, a quantum chromosome is a specific solution vector, in this case of real numbers, made up of several quantum genes. The number of genes of the quantum chromosome is determined by the required dimension of the solution vector. The explanation of this algorithm is as follows:

Algorithm 7 Real Quantum-Inspired Evolutionary Algorithm (RQIEA)

```
1  ProcedureRQIEA
2      t ← 1
3      initialize the quantum chromosomes Q(t) with N
       individuals with G genes
4      while t < T do
5          create the PDF's using the quantum individuals
6          E(t) ← generete classical population observing
           Q(t) and using CDF's
7          if t = 1 then
8              C(t) ← E(t)
9          else
10             E(t) ← Crossover between E(t) and C(t)
11             evaluate E(t)
12             C(t) ← K best individuals from
               [E(t) + C(t)]
13         end
14         with the N best individuals from C(t)
15             Q(t + 1) ← apply translate operation to Q(t)
16             Q(t + 1) ← apply resize operation to
               Q(t + 1)
17             t ← t + 1
18         end
19     end
20 End_RQIEA
```

1) At the start, all the genes in the quantum chromosome are initialized randomly, considering for each gene that its value must be within the range of allowable values for that dimension. The idea is to set the value of each gene within the allowable values for the corresponding dimension. For example, if we know that the allowable values for the dimension j on the solution vector are $[-80, 80]$, then the quantum gene i at dimension j is $g_{ij} = (p_{ij}, 160)$; the mean of the squared pulse is initialized randomly selecting a value within this range, say -40, then $g_{ij} = (-40, 160)$. At the initialization, the squared pulse does not need to be completed within the allowable range for a dimension because the algorithm will systematically adjust for this as it executes. The pulse's height from a gene j in the quantum chromosome i, h_{ij}, is calculated using (16)

$$h_{ij} = \frac{1/\sigma_{ij}}{N}. \tag{16}$$

Algorithm 8 Procedure Crossover of the RQIEA

```
1  Procedurecrossover
2      for i ← 1 to K do
3          select individual e_i from E(t)
4          select individual c_i from C(t)
5          for j ← 1 to G do
6              r ← choose random number in [0.1)
7              if r < ξ then
8                  e'_ij ← e_ij
9              else
10                 e'_ij ← c_ij
11             end
12         end
13     end
14 End_crossover
```

2) The **while** loop will iterate the steps within it during T generations, t is the generation counter. Inside the loop are the next steps:
   a) The interference between the quantum individuals is obtained by summing up all members of $Q(t)$; i.e., the quantum population. This process generates a probability density function (PDF) that ensures that the area under this PDF is 1. For the gene j on iteration t the PDF is given by (17), where $\bar{g}_{ij}$ symbolizes the phenotype and it is composed by the pair (g_{ij}, h_{ij}),

$$PDF_j(t) = \sum_{i}^{N} \bar{g}_{ij}. \tag{17}$$

   b) A classical population $E(t)$ is generated by observing $Q(t)$ and using the cumulative distribution function CDF. First, to obtain an observation it is necessary to calculate the cumulative distribution function (CDF) of a gene using (18); L_j^i and L_j^s are the upper and lower limit of function PDF_j. Individual q_i's are formed by N genes, hence, it is mandatory to construct a CDF for each gene considering the whole population,

$$CDF_j(x) = \int_{L_j^i}^{L_j^S} .PDF_j(x)dx \tag{18}$$

   The value of a gene in the classical population is generated using (19), where r is a random number in the interval $[0, 1]$. With this process, a classic temporary population $E(t)$ with K individuals is generated,

$$x = CDF^{-1}(r). \tag{19}$$

   c) In the first generation (line 8), an exact copy $C(t)$ of $E(t)$ is created.
   d) For all cases after the first generation, a crossover operation among the individuals of $E(t)$ and $C(t)$

is applied. Algorithm 8 depicts the crossover operator.

e) The population $E(t)$ is evaluated.

f) The new classical population $C(t)$ is formed by selecting the K best individuals from $E(t) \bigcup C(t)$.

g) The N best individual from $C(t)$ are used to update the quantum population $Q(T)$. This process consists of two operations named "translate" (line 15) and "resize" (line 16), and it is achieved in two steps:

i) The translate operation modifies the center ρ of the quantum genes. The procedure is achieved by making the gene's mean value ρ_{ij} of the center of the $j-th$ gene of the $i-th$ quantum individual in $Q(t)$ equal to the corresponding values in the classical individual c_{ij}; mathematically we have that,

$$\rho_{ij} = c_{ij}. \tag{20}$$

ii) The resize operation updates the corresponding width value of the $j-th$ gene of the $i-th$ quantum individual in $Q(t)$. The operation is applied to all the quantum genes for all the individuals; this is done to vary the exploration and exploitation abilities of the quantum search algorithm. The heuristic to decide if the new width σ_{ij} should be shrunk or enlarged is "1/5th" (Rechenberg rule [44]); this rule says that if less than 20% of the actual generation has improved, then the gene width is reduced to try to improve the local search. If the rate is precisely 20%, no changes are made; otherwise, the width is enlarged to try to improve the global search. The rule can be expressed mathematically using (21),

$$\sigma_{ij} = \begin{cases} \sigma_{ij} \cdot \delta & \varphi < 1/5 \\ \sigma_{ij} & \varphi = 1/5 \\ \sigma_{ij}/\delta & \varphi > 1/5. \end{cases} \tag{21}$$

The crossover operator depicted in Algorithm 8 uses a K number of individuals in the classical population $C(t)$, and G as the number of genes in each individual. The parameter ξ is the crossover rate; if $\xi = 1$ all the genes of the created individual will be copied to the offspring. A rate of $\xi = 0$ will not modify the individuals.

The RQIEA was tested in [32] using four benchmark functions; two are unimodal functions (Sphere and Schwefel 2.21), and the others are multimodal (Griewank and Ackley 1). The results were compared against the Stochastic Genetic Algorithm (StGA) [66], Fast Evolutionary Programming (FEP) [73], Fast Evolutionary Strategy (FES) [74], and Particle Swarm Optimization (PSO) [3]. The reported results were that the RQIEA outperformed the FEP, FES, PSO with fewer evaluations, and the StGA with the same number of evaluated functions, numerically.

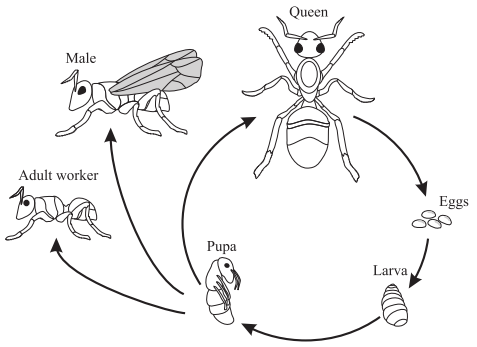

FIGURE 3. Life cycle of ants.

D. QUANTUM INSPIRED ACROMYRMEX EVOLUTIONARY ALGORITHM

The quantum-inspired *Acromyrmex* evolutionary algorithm (QIAEA) proposed by [48] was motivated by the colony behavior of the Atta and Acromyrmex, also known as leafcutter ants. This paradigm is different from the ant colony optimization (ACO) metaheuristics proposed by [15] and [63]. In Fig. 3, the life cycle of ants is shown; it consists of four stages: eggs, larva, pupa, and adult. Fertilized eggs produce female ants, which eventually become queens, workers, or soldiers. Unfertilized eggs produce male ants. The workers and the soldiers are sterile. Typically, these advanced social insects leave in nests, and they have well-organized colonies form by the queen $\mathcal{Q}$, males $\mathcal{M}$'s, workers and soldiers $\mathcal{S}$, and sometimes virgin queens $\mathcal{V}$. Depending on the species, there are slight variations in their behavior. Queens are the only ants with reproductive capabilities.

This algorithm was defined as follows,

- $\mathcal{P}_{\downarrow}$ defines an ant population, which consists in a set of N individuals (ants) p_i sorted in descending order; i.e., the best individuals are first.
- c_i to denote the chromosome representing a colony member of $\mathcal{C}$.
- $p_i = c_i \cup f_i$ is a conventional individual containing its fitness value f.
- The population of individuals $\mathscr{P}_{\downarrow}$ is described as,

$$\mathscr{P}_{\downarrow} = \{p_i : i \in N, p_i \geq p_{i+1}\} = \bigcup_{\substack{i \in N, \\ p_i \geq p_{i+1}}} p_i. \tag{22}$$

- The fitness function $\mathscr{F}$ assigns the corresponding fitness value f to each element of $\mathscr{P}$; i.e., $\mathscr{F} = \bigcup_{i \in N} f_i$, and $\mathcal{F} : \mathscr{P}_{\downarrow} \rightarrow \mathscr{F}_{\downarrow}$.
- The QIAEA is formally defined by five main elements: queens $\mathscr{Q}$, virgin queens $\mathscr{V}$, males $\mathscr{M}$, and the rest of the ant population $\mathscr{R}$.
- A particular implementation for the QIAEA with one queen is $\mathcal{P}_{\downarrow} = \{\mathcal{Q}_1, \mathcal{M}^{l}_{k=2}, \mathcal{R}^{N}_{l=l+1}, \mathcal{F}^{N}_{i}\}$. N, k, and l, are numbers that defines the quanty of each kind of ants.
- The algorithms use the symbol $\mathcal{O}$ to indicate quantum measurements.

- $\mathcal{O}_{SQM}$ means single qubit measurement. This procedure is very similar to the procedure shown in Algorithm 2 of the GQA; however, in $\mathcal{O}_{SQM}$ the algorithm specifies that random numbers must be generated using a uniform probability distribution.
- $\mathcal{O}_{FM}$ means full-measurement; i.e., it used to measure the whole quantum chromosome. This procedure is given in Algorithm 11.

The pseudocode of QIAEA is shown in Algorithm 9, next, a brief explanation of the main steps is given:

1) The iteration counter t is set to 0.
2) A binary ant colony $\mathcal{C}$ of size N is initialized in lines 3-8 as follows,
   a) The queen Q_0 is set in the zero state ($s_0 = 1$).
   b) Q_0 is set in the superposition state (Q_H) by applying the Hadamard gate to Q_0. The probability of the outcomes to be zero or one will be 50-50.
   c) The binary chromosome c_i is obtained by performing the full measurement of Q_H.
3) An ant (colony member) p_i is formed by its corresponding chromosome c_i and fitness f_i. In lines 9-11, a population $\mathcal{P}$ of p_i binary ants is created.
4) The population $\mathcal{P}$ is sorted in descending order; now, the fittest individual is p_1; hence the queen is $\mathcal{Q}_1$.
5) $\mathcal{P}$ is divided in two subpopulations, a queen $\mathcal{Q}_1$, and a set of males $\mathcal{M}^l_{k=2}$ of size $l-1$.
6) Best individuals are saved in $\mathcal{B}^j_1$, where j indicates the number of individuals to be saved.
7) The repeat-until loop (lines 16-28) will repeat the next steps until the termination criterion is achieved.
   a) The iteration counter is incremented.
   b) New colony members' chromosomes are created using only the Queen $\mathcal{Q}_1$ and $\mathcal{M}^l_{k=2}$ chromosomes excluding their fitness values; i.e., $\mathcal{Q}_1 \backslash f_i$ and $\mathcal{M}^l_{k=2} \backslash f^l_k$. The crossover algorithm is invoked $N-1$ times to achieve recombination of the queen, considering all the males in $\mathcal{P}$.
   c) In line 22, the new individuals p_i are created by adding to the chromosomes the fitness values.
   d) In line 24, the population $\mathcal{P}$ is sorted in descending order.
   e) In line 25, the queen $\mathcal{Q}_1$ is chosen.
   f) In line 26, the set of males $\mathcal{M}^l_{k=2}$ of size $l-1$ is chosen.
   g) Best individuals are saved in $\mathcal{B}^j_1$, where j indicates the number of individuals to be saved.

Algorithm 10 shows the crossover procedure. The input parameters of this algorithm are the binary chromosomes of the queen and male. It also uses the parameter a to indicate the number of qubits of the quantum chromosome of the queen that will be changed. First, the algorithm sets a quantum chromosome Q_0 that can be in any state. The state s_q is determined with the binary queen, and it is used to set Q_0 in the state that must be. In line 5, a random number in the range 0 to $size(q-a)$ is generated, and the value is assigned to r. The random number r is used to determine the point position to start the $r+a$ comparisons in the quantum chromosome to apply quantum operations to the corresponding qubits (alleles). In other words, we want to compare substrings of the queen and the selected male, bit by bit. When a coincidence exist, a Hadamard operation is applied to the corresponding bit; otherwise, the quantum X-gate is applied to the quantum allele.

Algorithm 9 The Quantum Inspired Acromyrmex Evolutionary Algorithm

Data: Quantum chromosome $Q_0 = s_0|0\ldots000\rangle + \cdots + s_{2^n-1}|1\ldots111\rangle$, n is the number of qubits, and s is the probability for every state
Result: $\mathcal{Q}_1(t)$

1 **ProcedureQIAEA**
2 $t \longleftarrow 0$
3 **for** $i \leftarrow 1$ **to** N **do**
4 $s_0 \leftarrow 1$
5 $Q_H \longleftarrow H(Q_0)$
6 $Q \longleftarrow \mathcal{O}_{FM}(Q_H)$
7 $c_i(t) \longleftarrow Q$
8 **end**
9 **for** $i \leftarrow 1$ **to** N **do**
10 $p_i(t) \longleftarrow c_i(t) \bigcup f_i(c_i)$
11 **end**
12 $\mathcal{P}_{\downarrow}(t) \longleftarrow \mathcal{P}(t)$
13 $\mathcal{Q}_1(t) = p_1(t)$
14 $\mathcal{M}^l_{k=2}(t) = [p_2(t), \cdots, p_l(t)]$
15 $\mathcal{B}^j_1 = [p_1(t), \cdots, p_j(t)]$
16 **repeat**
17 $t \leftarrow t+1$
18 **for** $\iota \leftarrow j$ **to** N **do**
19 $c_\iota(t) \leftarrow$`Crossover`$(\mathcal{Q}_1 \backslash f_1, \mathcal{M}^l_k \backslash f^l_k)$
20 **end**
21 **for** $i \leftarrow 1$ **to** N **do**
22 $p_i(t) \longleftarrow c_i(t) \bigcup f_i(c_i)$
23 **end**
24 $\mathcal{P}_{\downarrow}(t) \leftarrow \mathcal{P}(t)$
25 $\mathcal{Q}_1(t) \longleftarrow p_1(t)$
26 $\mathcal{M}^l_{k=2}(t) = [p_2(t), \cdots, p_l(t)]$
27 $\mathcal{B}^j_1 = [p_1(t), \cdots, p_j(t)]$
28 **until** *stop criteria*
29 A**End_QIAEA**

E. OTHER QUANTUM BASED EVOLUTIONARY ALGORITHMS

In [67], a modification to the GQA was proposed, and the new algorithm was named AQGA. It consisted in auto-adjust the rotation angle θ_t using $\theta_t = \theta_{max} - (\theta_{max} - \theta_{min}) * t/t_{max}$. In the AQGA, the value of $\theta_t \in \{\theta_{min}, \theta_{max}\}$ depends on the current

Algorithm 10 Crossover Algorithm. Crossover(q, m)

Data: q: queen, m: male, a: number of qubits to swap
Result: c_{new}: new chromosome

ProcedureCrossover
  $Q_0 = s_0|0\ldots000\rangle + \cdots + s_{2^n-1}|1\ldots111\rangle$
  $s_q \leftarrow 1$
  $r \longleftarrow \mathcal{N}(0, \text{sizeof}(q-a))$
  for $\kappa \leftarrow r$ **to** $r+a$ **do**
    if $q[\kappa] == m[\kappa]$ **then**
      $Q_0 \leftarrow H|\kappa\rangle$
    else
      $Q_0 \leftarrow X|\kappa\rangle$
    end
  end
  $c_{new} \leftarrow \mathcal{O}_{FM}(Q_0)$
End_Crossover

Algorithm 11 Full Measurement of a Quantum Chromosome (FM)

Data: qc: a quantum chromosome, nq: number of qubits to measure
Result: c_{new}: new chromosome

ProcedureFM
  for $\iota \leftarrow 0$ **to** $nq-1$ **do**
    $c_{new}[\iota] \leftarrow \mathcal{O}_{SQM}(qc[\iota])$
  end
End_FM

generation $t \in \{1, t_{max}\}$, initializing in θ_{max} and decreasing over time to θ_{min}. In [25], the first parallel programmed version of a quantum genetic algorithm (QGA) named parallel QGA (PQGA) appeared. It proved to be superior to the QGA optimizing the knapsack problem. In [7], the adaptive QIAG was proposed. It uses strategies to choose the best operators parameter. The algorithm was tested with the knapsack problem proving its superiority to the QIGA. Other quantum evolutionary works focused on solving combinatorial problems that demonstrate to be superior to the classical genetic algorithm GA are [12], [23], [24]. In [42], a quantum evolutionary clustering algorithm based on the watershed for texture image and SAR segmentation, was proposed. In this algorithm, the original image is divided into small pieces, and the quantum algorithm searches the optimal clustering center. In [40], the hybrid quantum-inspired genetic algorithm (HQGA) was proposed to solve the multiobjective flow shop scheduling problem (FSSP) [20]; which is an NP-hard combinatorial optimization problem. The HQGA fused the QGA [23] and a permutation-based genetic algorithm to obtain an integrated framework with the best of each algorithm.

IV. QUANTUM SWARM EVOLUTIONARY ALGORITHM

Swarm intelligence studies the collective behavior of natural and artificial systems that work collectively using decentralized control and self-organization. It is a branch of artificial intelligence. Some natural system examples studied by swarm intelligence are colonies of bees, ants, termites, a flock of birds, school of fish, fireflies, bats, and others. These swarm systems have inspired many algorithms to mimic their collective intelligence. The original proposals were developed for classical computing based on bits. At present, these swarm methods have been rethought inspired in quantum mechanics producing new algorithms to be used in classical computers. Next, we describe a selection of these swarm quantum-inspired algorithms.

A. QUANTUM-INSPIRED PARTICLE SWARM OPTIMIZATION

The quantum swarm evolutionary algorithm (QSwE) [68] is based on the QEA and in the particle swarm optimization method (PSO). The QSwE uses a different Q-bit expression called quantum angle as well as an improved PSO method to update quantum angles automatically. This method was tested with the 0-1 knapsack problem and the traveling salesman problem (TSP).

1) PARTICLE SWARM OPTIMIZATION

Artificial swarm intelligence is inspired by the study of colonies or swarms of social organisms. Its objective is to model the simple conduct of individuals and the local iterations with neighboring individuals and the environment. Particle swarm optimization (PSO) is an efficient and popular stochastic optimization approach that models the social behavior of birds within a flock [36], [62]. A PSO system is a population-based search method where individuals are referred to as particles that are flown through a hyperdimensional search space grouped into a swarm. A PSO algorithm maintains a set of particles named the swarm, and each particle is a potential solution. Generally, if $x_i(t)$ denotes the position of a particle i at time t, the position of the particle i is changed by a velocity term $v_i(t)$ added to the current position, according to (23). The velocity vector leads to the optimization process. It reflects the experimental knowledge, usually referred to as the *cognitive component*, and is proportional to the distance of the particle from its own best position (*personal best position*) found since the first time step. The socially exchanged information from the particle's neighborhood is the *social component* of the velocity equation,

$$\mathbf{x}_i(t+1) = \mathbf{x}_i(t) + \mathbf{v}_i(t+1). \tag{23}$$

The first two PSO developed algorithms were the global best (*gbest*) and the local best (*lbest*) PSO. In the *gbest* PSO, the neighborhood for each particle is the entire swarm, and the social network reflects a star topology. The *lbest* PSO uses a ring social network topology where smaller neighborhoods are defined for each particle.

2) THE QUANTUM SWARM EVOLUTIONARY ALGORITHM

The quantum swarm evolutionary algorithm (QSwE) [68] uses the *gbest* PSO where the velocity of a particle i is

calculated as shown next,

$$v_{ij}(t+1) = v_{ij}(t) + c_1 r_{1j}(t)[y_{ij}(t) - x_{ij}(t)] + c_2 r_{2j}(t)[\hat{y}_j(t) - x_{ij}(t)]; \quad (24)$$

where the personal best position y_i is the best position that the particle i has visited since the first time step, the best position found in the swarm is $\hat{y}(t)$, the velocity of the particle i in dimension j, $j = 1, \ldots, \infty$, at time t is denoted by v_{ij}, $x_{ij}(t)$ indicates the position of the particle i in dimension j at time t, c_1 and c_2 are positive acceleration constants that are used to scale the contribution of the cognitive and social components, respectively. The random values $r_{1j}(t)$ and $r_{2j}(t)$ introduce stochastic elements to the algorithm, they are in the range [0, 1], sampled from uniform distribution.

In the QSwE, a quantum particle is an array of quantum angles. Hence to use the PSO to update a qubit automatically, we need the following definition: "A quantum angle is an arbitrary angle θ, and a qubit is presented as $[\theta]$".

Therefore, $[\theta] = [\sin(\theta) \ \cos(\theta)]^T$ is equivalent to the original qubit, since it satisfies that $|\sin(\theta)^2 + \cos(\theta)^2| = 1$. Then, the equivalent of the quantum chromosome $\mathbf{q}$ of the QEA for the PSO is $\mathbf{q} = [[\theta_1] \mid [\theta_2] \ \ldots \ [\theta_m]]$.

Algorithm 12 Procedure Make for the QSwE, and the QIFAPSO

```
Procedure make(x)
    j ← 0
    while i < m do
        j ← i + 1
        i ← 0
        while j < k do
            i ← j + 1
            if random[0, 1] > |cos(θ_ij)|^2 then
                x_i ← 1
            else
                x_i ← 0
            end
        end
    end
End_Make
```

Using the definition of quantum particle $\mathbf{q}$. A quantum swarm is defined as $Q(t) = \{\mathbf{q}_1^t, \mathbf{q}_2^t, \ldots, \mathbf{q}_n^t\}$, where $\mathbf{q}_i^t = [[\theta_{i1}^t] \mid [\theta_{i2}^t] \ \ldots \ [\theta_{im}^t]]$. The QSwE uses similar pseudocode of the QEA shown in Algorithm 5, the differences are in the make and update procedures, they changed to work with quantum particles instead of the quantum individuals of the QEA.

In this **make** procedure version (Algorithm 12), each binary particle x_{ij}^t might have a value of 0 or 1 in $P(t)$ which is achieved by observing the states of $Q(t)$ and using $|\cos(\theta_{ij})|^2$ or $|\sin(\theta_{ij}^2)|$.

The **update** procedure was reformulated to use velocities and positions instead of using the rotation quantum gate $U(\theta)$ that calculates $[\theta_i' = \theta_i + \xi(\Delta\theta_i)]$. In the QSwE a velocity formula very similar to the Clerk proposal was used [8], [9]. The update procedure of the QSwE is given in Algorithm 13, it uses the next equations,

Algorithm 13 Procedure Update of the QSwE

```
Procedure update(q)
    i ← 0
    j ← 0
    while i < m do
        i ← i + 1
        while j < k do
            j ← j + 1;
            v_ij^{t+1} = χ[w v_ij^t + c_1 rand()[θ_ij^t(pbest) − θ_ij^t]] + c_2 rand()[θ_j^t(gbest) − θ_ij^t]
            θ_ij^{t+1} = θ_ij^t + v_ij^{t+1}
        end
    end
    q=q'
End_update
```

$$v_{ij}^{t+1} = \chi[w\, v_{ij}^t + c_1 r_1[\theta_{ij}^t(pbest) - \theta_{ij}^t] + c_2 r_2[\theta_j^t(gbest) - \theta_{ij}^t] \quad (25)$$

$$\theta_{ij}^{t+1} = \theta_{ij}^t + v_{ij}^{t+1}. \quad (26)$$

In the mentioned work, the following values were used: $\chi = 0.99$, $w = 0.7298$, $c_1 = 1.42$, and $c_2 = 1.57$; these values satisfies the convergence conditions of the particles $w > (c_1 + c_2)/2 - 1$. Because $c_2 > c_1$ the particles will converge faster to the global optimal position in the swarm than the local optimal position of each particle, therefore, the QSwE based on the PSO has global searching property [29].

The QSwE was tested solving the 0-1 knapsack problem and the TSP; in the first case, the same **repair** procedure described by Algorithm 4 was used, for the second case, a new **repair** procedure was proposed.

3) THE QUANTUM-INSPIRED FIREFLY ALGORITHM WITH PSO

Motivated by the natural behavior of the firefly (FF) insect, in [61], the Firefly Algorithm (FA) for global optimization was proposed [72]. It is based on the behavior and flashing patterns of FFs. The main idealized rules are:

- One FF can attract other FFs regardless of their gender because artificial fireflies are unisex. In nature, FFs send signals to the opposite sex; however, the idealized mathematical model considers unisex FFs.
- The attractiveness of the FFs is correlated to the brightness, and they increase as the distance decreases. Considering two FFs, a brighter FF will attract the less bright one. If there no exist brighter one, the existing will move randomly.

- The brightness of a FF is determined by the landscape of the objective function.

In the FA, there are two essential issues: the light intensity variation and the mathematical formulation of the attractiveness. For simplicity, it is assumed that the attractiveness of the FF is proportional to its brightness. A different FF perceives the brightness degree and is inversely proportional to the distance. The objective function to optimize determines the brightness. For example, for a maximizing problem, the brightness I at a particular position $\mathbf{x}$ of a FF can be selected as $I(\mathbf{x}) \propto f(\mathbf{x})$.

Basically, the light intensity $I_b(r)$ varies according to

$$I_b(r) = \frac{I_s}{r^2}, \tag{27}$$

where I_s is the intensity at the source and r is the distance. Considering a medium with fixed light absorption coefficient γ, and that I_0 is the original light intensity, then the light intensity is given by

$$I(r) = I_0 e^{-\gamma r}. \tag{28}$$

The attractiveness β of a FF is proportional to $I(r)$, then it is defined by

$$\beta = \beta_0 e^{-\gamma r^2}, \tag{29}$$

where the attractiveness at $r = 0$ is denoted by β_0. The exponential function can be calculated faster using the approximation $1/(1 + r^2)$; then (29) can be rewritten as

$$\beta = \frac{\beta_0}{1 + \gamma r^2}. \tag{30}$$

The Cartesian distance between two FFs i and j at $\mathbf{x}_i$ is calculated with

$$\mathbf{r}_{ij} = ||\mathbf{x}_i - \mathbf{x}_j||. \tag{31}$$

The movement of a FF i is attracted to another more attractive FF is given by,

$$\mathbf{x}_i = \mathbf{x}_i + \beta_0 \, e^{-\gamma r_{ij}^2}(\mathbf{x}_j - \mathbf{x}_i) + \alpha\epsilon_i; \tag{32}$$

in the second term, the attractiveness and the distance give the attraction factor; in the third term α is a random parameter, ϵ_i was defined as a vector of random numbers drawn from uniform or Gaussian distribution, in the simplest form ϵ_i is replaced by $rand - 0.5$. Most of the implementations use $\beta_0 = 1$ and $\alpha \in [0, 1]$.

The attractiveness given by (29) and in (30) defines a characteristic distance $\gamma = 1/\sqrt{\Gamma}$ over which the attractiveness changes significantly from β_0 to $\beta_0 \, e^{-1}$ in the case of (29), or $\beta_0/2$ in the case of (30). It is practical an attractive function $\beta(r)$ that can be any monotonically decreasing function, such as

$$\beta(r) = \beta_0 e^{-\gamma r^m} (m \geq 1). \tag{33}$$

For a fixed value of γ, the characteristic length is

$$\Gamma = \gamma^{-1/m} \rightarrow 1m \rightarrow \infty. \tag{34}$$

Conversely, for a known length scale, Γ in an optimization problem, γ can be used as an initial value, it is given by,

$$\gamma = \frac{1}{\Gamma^m}. \tag{35}$$

The parameter γ characterizes the variations in attractiveness, and it is of crucial importance to determine the speed of convergence. Theoretically, $\gamma \in [0, \infty)$; in practice, γ is obtained using the characteristic length Γ of the system to be optimized.

Algorithm 14 Procedure of the QIFAPSO

```
Procedureqifapso
    initialize quantum fireflies
    make quantum fireflies by quantum measure
    evaluate the fitness of binary fireflies
    if ¬ stop then
        if Bf_i < Bf_j then
            apply FFA formulas to update the quantum
            firefly solution Qf_i
        else
            apply the PSO formulas to update the
            quantum firefly solution Qf_i
        end
        make binary firefly solution Bf_i by quantum
        measure from quantum firefly solution Qf_i
        evaluate the fitness and update the brightness of
        binary firefly solution Bf_i
    end
    store the current best binary firefly solution
End_qifapso
```

The quantum-inspired firefly algorithm with the PSO (QIFAPSO) was proposed in [80]. As the name suggests, the QIFAPSO uses both metaheuristics; therefore, they use a similar quantum coding scheme.

In the binary representation, each FF solution Bf_i (the $i-th$ FF in a binary population) is coded using a vector of length m, which is the size of the solution; hence, $Bf_i = (Bf_{i1}, \ldots, Bf_{i2})$. As an example, $Bf_i = (1, 0, 0, 1)$ represents the solution vector (1001). A population of n FFs at iteration t is denoted by $BF(t) = \{Bf_1^t, \ldots, Bf_n^t\}$ for $1 \leq i \leq n$.

On the other hand, in the quantum representation, a quantum bit is defined as $[\theta] = [\cos(\theta) \;\; \sin(\theta)]^T$ where the probability amplitude satisfies $|\cos(\theta)^2 + \sin(\theta)^2| = 1$, where the angle $\theta \in [0, \frac{\pi}{2}]$. A quantum FF solution Qf_i is a vector of qubtis, in concordance with the binary solutions, each FF solution Qf_i (the $i - th$ FF in a quantum population) is associated with the vector of angles $\Theta_i = (\theta_{i1}, \ldots, \theta_{im})$ of length m. At iteration t, each quantum FF Qf_i is a set of quantum bits, given by

$$Qf_i^t = \begin{bmatrix} \cos(\theta_{i1}^0) & | & \cos(\theta_{i2}^0) & | & \cdots & | & \cos(\theta_{im}^0) \\ \sin(\theta_{i1}^0) & | & \sin(\theta_{i2}^0) & | & \cdots & | & \sin(\theta_{im}^0) \end{bmatrix}; \tag{36}$$

where $1 \leq i \leq n$, n is the number of FF in the population, and m is the length of each quantum FF solution Qf_i.

The pseudocode of this proposal is depicted in Algorithm 14. The decision point break about using the FA or the PSO is given in line 6, here the brightness of the binary FFs Bf_i and Bf_j are compared as follows:

If the value of Bf_i is smaller than Bf_j then line 7 is executed, and the formulas of the FFA (line 7) shown in (37) are used to update the quantum FF solution Qf_i,

$$\theta_{ik}^{t+1} = \begin{cases} \theta_{ik}^{t} + \beta_0 e^{-\gamma r_{ij}^2}(\theta_{ik}^{t} - \theta_{ik}^{t}) + \alpha_t \epsilon_{ik}^{t} & \text{if } Bf_{ik}^{t} \neq Bf_{jk}^{t} \\ \theta_{ik}^{t} & \text{otherwise.} \end{cases} \tag{37}$$

In any other case, the line 9 is executed, where the formulas of the PSO (38) and (39) are used to update the quantum FF solution Qf_i,

$$\Delta\theta_{ik}^{t+1} = w\ \Delta\theta_{ik}^{t} + c_1\ r_1[\theta b_{ik}^{t} - \theta_{ik}^{t}] + c_2\ r_2[\theta g_{j}^{t}(k) - \theta_{ik}^{t}] \tag{38}$$

$$\theta_{ij}^{t+1} = \begin{cases} \theta_{ik}^{t} + \Delta\theta_{ik}^{t+1} & \text{if } Bf_{ik} \neq Fg_{k}^{t} \text{or} Bf_{ik} \neq Bf_{ik}^{t} \\ \theta_{ij}^{t} & \text{otherwise.} \end{cases} \tag{39}$$

The **make** procedure to obtain binary FF by observing the quantum ones is the same that the PSO uses (Algorithm 12).

The calculus of the distance depends on the kind of problem. In particular, the QIFAPSO algorithm, instead of using the distance formula (31), it uses a variant of the Hamming distance, which permits us to quantify the difference between two sequences of strings.

The QIFAPSO algorithm was evaluated using the multidimensional knapsack problem. It was compared with the QEA, QSwE, and the adaptive quantum-inspired differential evolution algorithm (ADQE).

B. OTHER QUANTUM SWARM-BASED ALGORITHMS

In 2004 appeared the first proposal of a quantum swarm optimization (QPSO) algorithm to solve a code division multiple access (CDMA) problem [71].

Bee colonies are another source of swarm inspiration metaheuristics where the motivation is to model the intelligent behavior of natural honeybees. In nature, the bee swarm searches for the food collectively by three classes of honeybees, the employed, onlookers, and scout bees. The employed bees that exploit the food source take the information about food source back to the hive to share it with the onlooker bees. The information is shared into the hive through a dance performed by the employed bees, and the dance is proportional to the food quality. The onlookers decide for a source; good food sources attract more onlooker bees. When a food source has been exploited fully, all the employed bees associated with this source abandon it becoming scout bees. Scout bees are responsible for searching for new sources of food [19]. This natural behavior was defined algorithmically for computer simulation for solving benchmark SO numerical optimization problems in [34]. Regarding the quantum-inspired version of artificial bee colonies, in [5], a metaheuristic named quantum-inspired artificial bee Colony (QABC) was proposed, and similarly to the original proposal, it was tested for solving benchmark functions, mainly the algorithm was tested with Sphere, Rosenbrock and the Grienwank function. According to the reported results, optimizing the mentioned functions, the QABC outperformed the QSEA and conventional genetic algorithm.

The fish movement in colonies inspired the artificial fish swarm algorithm (AFSA) [41], which shows intelligent social behavior while searches for food and escape from danger. Artificial fish optimization methods are based on populations, and similarly than other global optimization methods; they can work without gradient information of the objective function; hence, they can solve complex nonlinear high dimensional problems. It is known that AFSAs require few parameters to be adjusted and have high convergence speeds. At present, there are many algorithm variations, all motivated by the same basic ideas of praying, swarming, following, movement, and leaping. An artificial fish (AF) is an entity that has all its data and behaviors encapsulated. The AF accepts information of environment by sense organs, this is using an emulation of a vision system, and react to the stimulus by the control of tail and fin. An AF mainly lives in its solution space and the states of other AFs. Typically an AFSA starts a set of randomly generated potential solutions, then applies the behavior operators to achieve the search for the optimum one interactively. The quantum artificial fish swarm algorithm (QAFSA) was proposed in [78]. This quantum version has demonstrated to improve results of AFSA when optimizing the benchmark functions such as the Schwefel F7, and others.

V. SOCIAL EVOLUTION ALGORITHM

Human interactions, knowledge, and opinions inspired social evolution algorithms [54]. The first algorithm that used these concepts is the human evolutionary model (HEM) [46]. It has a particular fuzzy inference system named mediative fuzzy logic (MFL) [47] that uses consensus knowledge from experts to infer the best parameters of the algorithm to guide evolution intelligently. HEM through MFL maintains a minimum population size to exploit search space regions very fast; when diversity is lost, the population size increases to recover the exploration ability until a new promising region is found. In [54], a mechanism to incorporate human bias in the selection of individuals through a second opinion when interactions between individuals are indecisive was proposed. Algorithm 15 shows the pseudocode of the quantum-inspired social evolution algorithm (QSE) [52]. The decision-making process is biased by subjective or objective knowledge. This algorithm was divided into the following three phases: Initialization, evaluation, interaction.

1) The **initialization phase** (lines 2-5) has the next three steps:

- Initialize the QSE control parameters, which are:
  - A set of best individuals $B(t)$ to interact with $P(t)$. At iteration 0, it is randomly initialized; i.e., $B(t=0)=\psi$.
  - MCN is the maximum number of iterations for which the interaction phase is executed.
  - IP is the number of individuals of the population.
  - NFC is a threshold value used to decide whether to interact objectively in the whole population or subjectively within its neighborhood.
  - SFL is the learning factor that represents a global bias threshold.
  - IDF is a threshold value used to represent whether on not an iteration is decisive.

2) The **evaluation phase** (lines 6 and 7) perform the next two activities:
   - The fitness of each individual is calculated though a fitness function. The fitness is normalized over the whole population, and the best individual is saved.
   - Add in $B(t)$ the best solution in $P(t)$. At the start (t=0), $B(t)$ is initialized to $global-best$.
3) In the **iteration phase** (lines 10-21), the probabilistic fitness nf of X_i is compared with a random value $R1$, if $nf < R1$ then individual has decided not interact. Otherwise, the *iterationNumber* parameter is incremented, the individual I is selected to interact with, the **Procedure-1** is executed, and the based on the quality of interactions is decide to take a second opinion executing **Procedure-2**, or not.
   - **Procedure-1** depicted by Algorithm 16 was developed to model subjective/objective bias. Objective bias is selected in line 2 when the condition ($I\_SOB < SOB$) is true. In this case, the $global-best$ individual is selected. The interaction is performed using a strategy similar to the QEA, where a rotation gate is used. For this algorithm, there are three angles, and they will behave according to Table 3 with $\Delta\theta_3$ and $\Delta\theta_5$ set between 0.08π and 0.1π and other $'\Delta\theta_i'$ is set between 0.001π and 0.1π. Subjective bias is selected when the condition ($I\_SOB < SOB$) is false. Here the condition $I\_SFL() < SLF$ is also checked. If it is true, a random solution from the list of the previous generation's best solution is selected. Otherwise, the $local-best$ individual in the current population is selected, and the interaction is performed using the quantum rotation gate with $'\Delta\theta_i'$ between 0.001π and 0.1π.
   - **Procedure-2** After performing the subjective iteration Algorithm 17 is executed.

Algorithm 15 Quantum Inspired Social Evolution Algorithm (QSE)

```
1  ProcedureQSE
2      t ← 0
3      initialize control parameters
4      initialize Q(t)
5      observe Q(t) to obtain P(t)
6      compute fitness and normalized fitness (nf) of
       individuals in P(t)
7      store the best solution among P(t) into global − best
       and its iteration count in B(t)
8      repeat
9          t ← t + 1
10         interactionNumber ← 0
11         foreach individual I ∈ P(t) do
12             R1 ← randomnumber[0, 1]
13             if nf > R1 then
14                 iterationNumber ←
                   iterationNumber + 1
15                 select individual I to interact with
16                 establish interaction based on
                   subjective/objective bias using
                   Procedure-1 for all subjective
                   interactions, evaluate quality of
                   interactions
17                 if quality of interaction is not
                   satisfactory then
18                     take second opinion using
                       Procedure-2
19                 end
20             end
21         end
22         observe Q(t) to obtain P(t)
23         compute the fitness and nf of each individual in
           P(t)
24         store the best solution among P(t) into
           global − best and its iteration count in B(t)
25     until t < MCN
26 EEnd_QSE
```

VI. MULTIOBEJTIVE QUANTUM-NATURE-INSPIRED ALGORITHMS

Single objective optimization (SOO) deals with only one objective function (OF), and the only goal is to find the optimal global solution. In contrast, multiobjective optimization (MOO) deals with more than one OF, and it has two goals. One goal is to find the optimal set of solutions in the Pareto front. The second goal is that the optimal set has a big diversity among solutions. Another difference is that SOO works only in the decision variables space that can be multidimensional; whereas MOO works in a multidimensional space formed by the OFs, it is commonly named the objective space.

The methods to solve MOO problems (MOOP) can be divided into classical and metaheuristic methods. The first

TABLE 3. Lookup table to find the appropriated rotation angle $\Delta\theta_i$.

					$s(\alpha_i\beta_i)$			
x_i	b_i	$f(\mathbf{x}) > f(\mathbf{b})$	Case	$\Delta\theta_i$	$\alpha_i\beta_i \leq 0$	$\alpha_i\beta_i \leq 0$	$\alpha_i = 0$	$\beta_i = 0$
0	0	false	θ_1	0	0	0	0	0
0	0	true	θ_2	0	0	0	0	0
0	1	false	θ_3	$\Delta\theta_3$	1	-1	$\pm$ 1	0
0	1	true	θ_4	$\Delta\theta_4$	-1	1	$\pm$1	0
1	0	false	θ_5	$\Delta\theta_5$	-1	1	$\pm$1	0
1	0	true	θ_6	$\Delta\theta_6$	1	-1	0	$\pm$1
1	1	false	θ_7	$\Delta\theta_7$	1	-1	0	$\pm$1
1	1	true	θ_8	$\Delta\theta_8$	1	-1	0	$\pm$1

Algorithm 16 QSE - Procedure 1

```
1  Procedure Procedure_1
2      if I_SOB < SOB then
3          Select the global − best identified till now
4          Interaction is performed using Random search
           based QEA rotation gate as described in Table 3
           with Δθ3 and Δθ5 set between 0.08π and 0.1π
           and other Δθi set between 0.0001π and 0.π.
5      else
6          if I_SFL() < SelectiveLearningFactor(SLF)
           then
7              Select random solution from the list of
               previous generation's best solutions
8          else
9              Select the local − best individual in the
               current population
10             Interaction is performed using quantum
               rotation gate with Δθi set between 0.001π
               and 0.1π
11         end
12     end
13 End_Procedure_1
```

Algorithm 17 QSE - Procedure 2

```
1  Procedure Procedure_2
2      if I_SOB > SOB then
3          Assign a random indecisive factor (I_IDF) for
           every individual's interaction
4      end
5      if I_IDF < IDF then
6          Interact with the global − best using quantum
           rotation gate with Δθi set between 0.0001π and
           0.1π
7      end
8  End_Procedure_2
```

approach uses classical single objective optimization methods to provide solutions for each OF and then use a preference-based method to establish a trade-off among the OF and provide a solution or a set of solutions. The metaheuristic approach use population-based methods, such as evolutionary algorithms or swarm optimization-based methods; these methods have the ability to find multiple optimal solutions at each iteration, improving the set of solutions in the Pareto front [13].

A. QUANTUM-INSPIRED MULTIOBJECTIVE EVOLUTIONARY ALGORITHM (QMEA)

The quantum-inspired multiobjective evolutionary algorithm (QMEA) was based on the QEA. It is an approach proposed to solve the 0-1 knapsack problem [37]. The main procedure of the QMEA is depicted in Algorithm 18, where lines 1-9 are the same as the GQA. Lines 10-14 are different, and they are explained next.

1) Line 10. The QMEA runs the fast nondominated sorting procedure described in Algorithm 19 [14]. For each solution p, we calculate a domination count n_p to keep track of the number of solutions that dominates the solution p, and a set of solutions S_p, which are dominated by p. All solutions that have the domination count as zero belongs to the first nondominated front. For each solution p with $n_p = 0$ of this first front, we visit each member q of S_p, and the domination count is reduced by one. By doing this, if for any member q the domination count n_p becomes zero, that member is saved into a separated list Q. These members are elements of the second nondominated front. Now, the above process is repeated with each member of Q, and the third front is identified.
2) Line 11. The crowding distance and sorting are achieved using Algorithm 20. It is used to maintain diversity within the population in each non-dominated frontier; for each individual, a density estimator named crowding distance is used. An estimate of the density of solutions neighboring a particular solution can be obtained by averaging the distance of two solutions on either side of the particular solution along each of the objectives. The parameter $i_{distance}$ is called the crowding distance, and its computation requires having the population sorted according to each OF value in ascending order. After that, an infinite distance value is assigned to the boundary solutions. The intermediate solutions are designated with a distance value equal to the absolute normalized difference in the function values of two adjacent solutions. The same process continues with the other OFs. The global crowding-distance

Algorithm 18 Quantum-Inspired Muliobjective Evolutionary Algorithm (QMEA)

```
1  ProcedureQMEA
2      t ← 0
3      initialize the population of quantum chromosomes Q(t)
4      make P(t) by observing Q(t) states
5      evaluate each binary solution in P(t)
6      while not termination-condition do
7          t ← t + 1
8          make P(t) by observing Q(t − 1) states
9          evaluate each binary solution in P(t)
10         run the fast nondominated sort algorithm for P(t) ⋃ P(t − 1)
11         calculate crowding distance and sort
12         P(t) is formed by the first N elements in the sorted pupulation 2N
13         Q(t) is classified into several groups
14         update Q(t) using quantum gates refer to best group
15     end
16 EEnd_QMEA
```

value is computed as the sum of individual distance values corresponding to each objective. Therefore, the procedure calculates the crowding distance of all solutions in a nondominated set $\mathcal{I}$. The $m-th$ OF value or the $i-th$ individual in the set $\mathcal{I}$ is denoted by $\mathcal{I}[i].m$. The parameters f_m^{min} and f_m^{max} are the minimum and maximum values of the $m-th$ OF. Once all the members in the set $\mathcal{I}$ were assigned with distance metric, the comparison of two solutions for their extent of proximity with another solution can be achieved.

3) Line 12. The new population $P(t)$ is form by selecting the first N elements from the sorted population of size $2N$ since $(P(t) \bigcup P(t-1))$.
4) Line 13. Q(t) is classified in several groups. The population is divided into several groups, $G_1, G_2, \ldots, G_n$, starting in the top front in the stored population $P(t)$. It is expected that G_1 be the best group since solution were selected using the crowding distance already sorted, hence G_1 is used to update the quantum individual of other gropus. The best solution **b** is in G_1. Quantum individual in groups of lower rank $(G_2, G_3, \ldots, G_n)$ are updated using the best group G_1. Because it is an elitist strategy, quantum individuals in G_1 are retained.
5) Line 14. **update** $Q(t)$ using Q-gates refered to best group; for example, the rotation gate is used to perturb the Q-bit. This is done instead using the classical crossover and mutation operations.

The QMEA incorporates concepts of multiobjective optimization (MO) based on the Pareto front, particularly the fast nondominated sorting procedure (Algorithm 19) and the crowding distance assignment procedure (Algorithm 20) used by the NSGA-II framework.

Algorithm 19 Fast Nondominated Sorted

```
1  Procedurefast-non-dominated-sort(P)
       Input: Population P
       Output: Pareto fronts F_1, ..., F_i
2      foreach p ∈ P do
3          S_p ← ∅
4          n_p ← 0
5          foreach q ∈ P do
6              if p ≺ q then
7                  S_p ← S_p ∪ {q}
8              else if q ≺ p then
9                  n_p ← n_p + 1
10         end
11         if n_p = 0 then
12             p_rank ← 1
13             F_1 ← F_1 ∪ {p}
14         end
15     end
16     i ← 1
17     while F_i ≠ ∅ do
18         P̂ ← ∅
19         foreach p ∈ F_1 do
20             foreach p̂ ∈ S_p do
21                 n_p̂ ← n_p̂ − 1
22                 if n_p̂ = 0 then
23                     p̂_rank ← i + 1
24                     P̂ ← P̂ ∪ {p̂}
25                 end
26             end
27         end
28         i = i + 1
29         F_i = P̂
30     end
31 End_fast-non-dominated-sort(P)
```

B. OTHER MULTIOBJECTIVE QUANTUM-INSPIRED ALGORITHMS

A multiobjective QPSO to solve combinatorial logic circuits to obtain the feasible design of circuits with the minimal number of gates was proposed in [49]. The QPSO was applied to solve successfully linear array antenna problems. The results were compared against an improved version of the PSO. Other applications of QPSO in different fields such as economic load dispatch, image watermarking, and automatic image annotation using feature selection, can be found in [27], [64], and [33].

VII. OTHER SOURCES OF INSPIRATION

This category is dedicated to those algorithms inspired by different sources of inspiration that were not covered in the other sections.

Algorithm 20 Procedure to Assign Crowding Distances

1 **Procedurecrowding_distance_assignment**
  Input: Set of individuals $\mathcal{I}$
2 $l \leftarrow |\mathcal{I}|$
3 **foreach** $i \in \mathcal{I}$ **do**
4 $\mathcal{I}[i]_{distancia} \leftarrow 0$
5 **end**
6 **foreach** *objetive m* **do**
7 $\mathcal{I} \leftarrow \text{sort}(\mathcal{I}, m)$
8 $\mathcal{I}[1]_{distance} \leftarrow \infty$
9 $\mathcal{I}[l]_{distance} \leftarrow \infty$
10 **for** $i = 2$ *to* $(l-1)$ **do**
11 $\mathcal{I}[i]_{distance} \leftarrow$ $\mathcal{I}[i]_{distance} + \frac{(\mathcal{I}[i+1].m - \mathcal{I}[i-1].m)}{(f_m^{max} - f_m^{min})}$
12 **end**
13 **end**
14 **End_crowding_distance_assignment**

A. QUANTUM-INSPIRED GRAVITATIONAL SEARCH ALGORITHM

The natural force of gravity inspires the gravitational search algorithm (GSA) introduced in [56], and it is a swarm optimization strategy. A binary-valued variant named binary gravitational search algorithm (BGSA) was published in [57]. The GSA uses Newton's law of gravity and motion. At present, there are several variants developed to improve the original proposal like [60], and [58]. The first quantum computing of the GSA named quantum-behaved gravitational search algorithm (QGSA) was proposed in [45].

In Algorithm 21, the pseudocode of the standard GSA is shown. Individuals in the population are named agents. In line 3, an array X_i is initialized with N agents x_i^d; where D is the dimension of the search space. In steps 5 to 7, a fitness value is assigned to all the population agents. The next process is repeated t number of iterations. Calculate the values of $G(t)$, $best(t)$, $worst(t)$, $M_i(t)$ related to the gravity, best and worst individuals in the population, and the mass of the i individual, respectively. As is shown in step 10, the gravitational constant is reduced by time. In step 15, the calculus of the total force in different directions is achieved. In step 16, the calculus of acceleration and velocity is achieved. In step 17, the position of the agents is updated. In lines 19-20, the fitness of the new agents is calculated. Then, the iteration counter is updated, and the process is repeated a t_{max} number of iterations.

In the GSA, similarly to Newtonian mechanics, an individual is described by its position and velocity vectors, x_i and v_i, respectively; they determine the trajectory of the individual. However, in quantum mechanics, the term trajectory is meaningless because according to the uncertainty principle, the position and velocity cannot be determined simultaneously. Therefore, now we will analyze the necessary changes

Algorithm 21 Standard GSA

1 **ProcedureGSA**
2 $t \leftarrow 0$ **for** *i=1 to N* **do**
3 $[X_i] \leftarrow x_i^D$
4 **endFor**
5 **for** *i=1 to N* **do**
6 $fit_i \leftarrow f_{obj}(x_i^D)$
7 **endFor**
8 **do**
9 **for** *i=1 to N* **do**
10 $G(t) \leftarrow G(G_0, t)$
11 $best(t) = \max_{j \in \{1,\ldots,s\}} \text{fit}_j(t)$
12 $worst(t) = \min_{j \in \{1,\ldots,s\}} \text{fit}_j(t)$
13 $q_i(t) = \frac{\text{fit}_i(t) - \text{worst(t)}}{\text{best(t)} - \text{worst(t)}}$
14 $M_i(t) = \frac{q_i(t)}{\sum_{j=1}^{s} q_j(t)}$
15 $F_i^d(t) = \sum_{j \in kbest, j \neq i} rand_j G_0(t) \frac{M_j(t) M_i(t)}{R_{ij}(t) + \epsilon} (x_j^d(t) - x_i^d(t))$
16 $a_i^d = \frac{F_i^d(t)}{M_i(t)} = \sum_{j \in kbest, j \neq i} rand_j G(t) \frac{M_j(t)}{R_{ij}(t) + \epsilon} (x_j^d(t) - x_i^d(t))$
  $v_i^d(t+1) = rand_i\, v_i^d + a_i^d$
17 $x_i^d(t+1) = x_i^d(t) + v_i^d(t+1)$
18 **endFor**
19 **for** *i=1 to N* **do**
20 $fit_i \leftarrow f_{obj}(x_i^D)$
21 **endFor**
22 $t \leftarrow t + 1$
23 **while** $(t < t_{max})$
24 **End_GSA**
25 while

made to Algorithm 21 to pass the GSA to the quantum world.

In quantum mechanics, the general time-dependent Schrödinger equation is given by,

$$j\hbar \frac{\partial}{\partial t} \psi(r, t) = H(r) \psi(r, t); \tag{40}$$

where $\hbar$ is Planck's constant, Ψ is the wave function, and $H(r)$ is the Hamiltonian operator given by,

$$H(r) = -\frac{\hbar^2}{2m} \nabla^2 + V(r); \tag{41}$$

where ∇^2 is the Laplacian operator, m is the mass of the agent, and $V(r)$ is the potential energy distribution.

In the Schrödinger equation, the wavefunction $\psi(r, t)$ is the unknown. Its amplitude squared, $|\psi|^2 = \text{Q}$, is a probability measure for the agent's motion. It allows computing the probability of finding an object in a particular region at a

specific time. $|\Psi|^2$ as the probability density function (PDF) should satisfy the following condition

$$\int_{-\infty}^{+\infty} |\psi|^2 dr = \int_{-\infty}^{+\infty} Q dr = 1. \tag{42}$$

Now, we consider that the GSA is a quantum system. We can assume that an individual mass is moving in a Delta potential in a feasible search space, where the center is the weighted average of all the best individuals (*kbest*). To simplify, we are studying an agent in one-dimensional space; therefore, the potential energy of the mass in the Delta potential well is,

$$V(r) = -\gamma\delta(r); \tag{43}$$

where the positive number γ is proportional to the depth of the potential well.

For this model, the Schrödinger equation is,

$$E\psi(r) = \left(-\frac{\hbar^2}{2m}\nabla^2 - \gamma\delta(r)\right)\psi(r). \tag{44}$$

For $X \neq 0$, we can write the above equation as,

$$\frac{d^2\psi}{dr^2} + \frac{2m}{\hbar^2}E\psi = 0. \tag{45}$$

The following boundaries are uses to prevent divergence of the agents,

$$|r| \to \infty \leftrightarrow \psi \to 0; \tag{46}$$

and equation (45) is calculated as shown next to satisfy the bound conditions,

$$\psi(r) = e^{\frac{\sqrt{-2mE}}{\hbar}|r|} \text{when} r \neq 0. \tag{47}$$

In the QGSA, the evaluation of the fitness value requires to know the precise information of the agent. However, $\psi(r)$ can only provide the PDF of the mass to appear at the desired position. So, we need to measure the position of the agent, which is done by collapsing the quantum state to a classical state. To this aim, it is necessary to simulate the measurement process using the Monte Carlo method to evaluate uncertainty. Hence, it is possible to generate a random number "rand" uniformly distributed between 0 and 1, in such a way that (47) can be simplified by substituting a random number instead of that equation. Therefore,

$$Q(r) = |\psi(r)|^2 = e^{\frac{2\sqrt{-2mE}}{\hbar}|r|} = rand. \tag{48}$$

By simple mathematics, we have:

$$-\frac{2\sqrt{-2mE}}{\hbar}|r| = \ln(rand) \tag{49}$$

$$|r| = \frac{\hbar}{2\sqrt{-2mE}} \ln\left(\frac{1}{rand}\right) \tag{50}$$

$$r = x - best = \pm\frac{\hbar}{2\sqrt{-2ME}} \ln(rand). \tag{51}$$

Thus, the accurate position of an agent is measured as follows,

$$x = \text{best} \pm \frac{\hbar}{2\sqrt{-2mE}} \ln(rand). \tag{52}$$

The mass (agent) move according to the next iterative equations, where the design parameter δ is called the contraction-expansion coefficient, *rand* and S are random parameters with uniform distribution probability in range [0, 1].

$$\begin{cases} X_i(t+1) = Best_i + \delta|Best_i - X_i(t)| \ln\left(\frac{1}{rand}\right) & \text{if } S \geq 0.5 \\ X_i(t+1) = Best_i - \delta|Best_i - X_i(t)| \ln\left(\frac{1}{rand}\right) & \text{if } S < 0.5. \end{cases} \tag{53}$$

In [6], a modified binary quantum-behaved gravitational search algorithm with differential mutation (MBQGSA-DM) was proposed; in general, this algorithm and the QGSA have similitudes; because the MBQGSA-DM also includes differential mutation. Its pseudocode is shown in Algorithm 22. Comparing this pseudocode against the pseudocode of the standard GSA shown in Algorithm 21, it can be noted that the most essential difference is that in the GSA the update the location of agents use velocity and position formulas, whereas in the quantum versions, based on the Heisenberg principle of uncertainty, it is impossible to evaluate both the velocity and the position simultaneously. Therefore, the quantum-inspired gravitational search algorithms only use the position formula in the update equation.

In Algorithm 22, $Xpbest_i(t)$ is the position with best fitness value of the i-th particle up to iteration t. $Xkbest_j(t)$ is the position of the j-th particle that belongs to the set of K particles having the best fitness at iteration t. The fitness of a particle i at iteration t is defined as the ratio (see line 6 of the Algorithm) of the intraclass distance ($Dintra_i^t$) given by the equation shown in line 7 to the interclass distance ($Dinter_i^t$) calculated as shown in line 8. The differential mutation factor F is defined as $F = \frac{t}{T}$ as shown in line 20.

VIII. ANALYSIS OF THE METHODS

The analysis of the revised quantum metaheuristics is not an easy task; there are many considerations to observe since all the reported results in the different works were achieved under different conditions, ranging from computer technology, programming languages, translation of algorithms to programs, kind of benchmark problems, reliability of results, interpretation of results, comparison among methods, metrics to report results, and others. All the mentioned conditions could produce different outcomes, even for the same algorithms.

The study of the different sources of inspiration initially conceived for classical computer implementation that evolved to the corresponding quantum version is important. However, there are hundreds of papers that have comparisons of these metaheuristics for solving different problems in practically all the scientific and technological fields. According to

Algorithm 22 MBQGSA-DM

ProcedureMBQGSA-DM

Create p particles and make randomized initialization of their n dimensional positions $X \in \{0, 1\}$. Total number of iterations = T. Iteration counter = t.

$t \leftarrow 1$

repeat

  for *i=1 to p* **do**

    Calculate fitness $fit_i^t = \frac{Dintra_i^t}{Dinter_i^t}$

    where $Dintra_i^t = \sum_c \sum_{u,r \in c, u \neq r} ||F_{u,i}^t - F_r, i^t||$

    and $Dinter_i^t = \sum_{c,c'} \sum_{u \in c, r \in c', c \neq c' \neq r} ||F_{u,i}^t - F_r, i^t||$

  endFor

  Calculate $Xgbest_i(t)$

  for *i=1 to p* **do**

    Find $Xpbest_i(t)$

    $Xmbest_i(t) = \frac{\sum_{j=1}^{p} \frac{1}{||Xi(t), Xj(t)||} X_j(t)}{\sum_{j=1}^{p} \frac{1}{||Xi(t), Xj(t)||}}$

    $Xbest_i(t) = \frac{c_1\ Xmbest_i(t) + c_2\ Xpbest_i(t) + c_3\ Xgbest(t)}{c_1 + c_2 + c_3}$

    Consider uniform random numbers $r \in [0, 1] and R \in [0, 1]$

    if $r < 0.5$ **then**

$$\begin{cases} X_i(t+1) = Xbest_i + \delta |Xbest_i - X_i(t)| \ln\left(\frac{1}{rand}\right) & \text{if} S \geq 0.5 \\ X_i(t+1) = Xbest_i - \delta |Xbest_i - X_i(t)| \ln\left(\frac{1}{rand}\right) & \text{if} S < 0.5 \end{cases}$$

    end

    else

      $F = \frac{t}{T}$

      $X_i(t+1) = X_j(t) + (1 - F) \times [X_j(t) - X_m(t)] + F \times [Xgbest(t) - X_j(t)]$

    end

  end

  for *d=1 to n* **do**

$$x_i^d(t+1) = \begin{cases} 1, & \frac{1}{1 + e^{-x_i^d(t+1)}} > rand \\ 0, & otherwise \end{cases}$$

  end

until $t = T$

SEnd_MBQGSA-DM

the reported results in the quantum versions of the mentioned metaheuristics, the results are at least as good as their classical version counterpart. In many cases, the quantum version outperforms the classical ones.

Since the concepts and principles to achieve the first quantum evolutionary metaheuristics to solve a combinatorial optimization problem [51] were settle, almost a quarter of a century has elapsed. In those days, there was not possible to develop and test algorithms for a real quantum computer such as the D-Wave or the IBM Q system. Hence the term ''quantum-inspired'' was coined to specify that any algorithm tagged with the term was developed using concepts from quantum physics for a classical computer as the target system.

At present, an important point to consider is which ones of the proposed metaheuristics are more suitable to be implemented in a real quantum computer with small changes, following the proposed algorithm. One of the most popular quantum computers is the IBM Q; this system uses the circuit model codification scheme to implement quantum algorithms. If we consider the circuit model as our target programming paradigm, **complexity** can be defined in terms of **width**, **size**, and **length** of the quantum circuit. Width is the total number of qubits of the circuit. Size is the total number of gates that the circuit uses. Length is the circuit depth; it refers to the longest path from the input to the output (when the measure is performed). Frequently, the length of the quantum circuit is the main indicator of the complexity of the quantum circuit. The mentioned complexity measures can be used to analyze and evaluate this subjective selection objectively about the possible translation of the reviewed algorithms to the circuit model for a real quantum computer.

TABLE 4. Benchmark functions.

Name	Formula	Global minimum	Range
1. Circle	$(x_1+5)^2+(x_2+10)^2$	$f(x^*)=0, x^*=(-5,10)$	[-15,15]
2. Schwefel	$418.9829d-\sum_{i=1}^{d} x_i \sin(\sqrt{\lvert x_i\rvert})$	$f(x^*)=0,$ $x^*=[420.9687,420.9687]$	[-500,500]
3. Rastrigin	$10d+\sum_{i=1}^{d}\left[x_i^2-10\cos(2\pi x_i)\right]$	$f(x^*)=0, x^*=(0,0)$	[-5.12,5.12]
4. Drop-Wave	$-\frac{1+\cos(12\sqrt{x_1^2+x_2^2})}{0.5(x_1^2+x_2^2)+2}$	$f(x^*)=-1, x^*=(0,0)$	[-5.12,5.12]
5. Levi No. 13	$\sin^2(3\pi x_1)+(x_1-1)^2[1+\sin^2(3\pi x_2)]+$ $(x_2-1)^2[1+\sin^2(2\pi x_2)]$	$f(x^*)=0, x^*=(1,1)$	[-10,10]
6. Schaffer No. 2	$0.5+\frac{sin^2(x_1^2-x_2^2)-0.5}{[1+0.001(x_1^2+x_2^2)]^2}$	$f(x^*)=0, x^*=(0,0)$	[-100,100]
7. Shubert	$\left(\sum_{i=1}^{5} i\cos((i+1)x_1+i)\right)\left(\sum_{i=1}^{5} i\cos((i+1)x_2+i)\right)$	$f(x^*)=-186.7309,$ $x^*=many$	[-10,10]
8. Price No. 2	$1+\sin^2(x_1)+\sin^2(x_2)-0.1\exp^{-x_1^2-x_2^2}$	$f(x^*)=0.9, x^*=(0,0)$	[-10,10]
9. Rosenbrook	$\sum_{i=1}^{d-1}\left[100(x_{i+1}-x_i^2)^2+(x_i-1)^2\right]$	$f(x^*)=0, x^*=(1,1)$	[-2.048,2.048]
10. Michalewicz	$-\sum_{i=1}^{d}\sin(x_i)\sin^{2m}\left(\frac{ix_i^2}{\pi}\right)$	$f(x^*)\approx-1.8013,$ $x^*=(2.20,1.57),$ $m=103$	[0,π]
11. Six-Hump Camel	$\left(4-2.1x_1^2+\frac{x_1^4}{3}\right)x_1^2+x_1x_2+\left(-4+4x_2^2\right)x_2^2$	$f(x^*)=-1.0316,$ $x^*=(0.0898,-0.7126),$ $x^*=(-0.0898,0.7126)$	[-3,3]
12. Holder	$-\left\lvert\sin(x_1)\cos(x_2)\exp\left(\left\lvert1-\frac{\sqrt{x_1^2+x_2^2}}{\pi}\right\rvert\right)\right\rvert$	$f(x^*)=-19.2085,$ $x^*=(\pm8.0550,\pm9.6645)$	[-10,10]
13. Trigonometric	$1+\sum_{i=1}^{d}8\sin^2\left[7(x_i-0.9)^2\right]+6\sin^2\left[14(x_i-0.9)^2\right]$ $+(x_i-0.9)^2$	$f(x^*)=1,$ $x^*=(0,0)$	[-500,500]
14. Cross-in-Tray	$-0.0001\left(\left\lvert\sin(x_1)\sin(x_2)\exp\left(\left\lvert100-\frac{\sqrt{x_1^2+x_2^2}}{\pi}\right\rvert\right)\right\rvert+1\right)^{0.1}$	$f(x^*)=-2.06261,$ $x^*=(\pm1.3491,\pm1.3491)$	[-10,10]
15. Griewank	$\sum_{i=1}^{d}\frac{x_i^2}{4000}-\prod_{i=1}^{d}\cos\left(\frac{x_i}{\sqrt{i}}\right)+1$	$f(x^*)=0, x^*=(0,0)$	[-600,600]

Considering the **width** as the first complexity measure. In the reviewed works, the metaheuristics coded the possible solutions using either quantum registers containing qubits, real numbers instead qubits, or the concept of quantum angle. The first parameter that we can use to exclude possible metaheuristics to be translated to the circuit model with few modifications to the original algorithm is the coding of the quantum-register. Hence, we will exclude those metaheuristics that use quantum registers using real numbers instead of qubits, and those that use the concept of quantum angle. The metaheuristics that offer an easy translation with few modifications from the original algorithm to the circuit model for being used in a real quantum computer are those that use quantum-registers coded with qubits, contrary to using quantum-angle or real numbers to form a kind of qubit out of its original mathematical definition.

Taking as an example, a continuous optimization problem. Its width is given by three things: the numeric range of decision variables, the numerical resolution to express values of decision variables, and the number of variables of the problem. Hence, if we have an optimization problem with two decision variables, and each variable can be represented satisfactorily using ten classical bits, we will require a classical register of 20 bits, which corresponds to 20 qubits in the quantum world. At present, the maximal number of qubits that handles IBM Q is 20; therefore, this is a limitation for most of the analyzed metaheuristics.

Concerning the **size** of the quantum circuit, the mentioned metaheuristic algorithms work with quantum register containing several qubits. It is desirable that the quantum circuit grows as a polynomial of size n^k with $k>0$. If the quantum circuit size grows exponentially, as functions like e^n, or 2^n, we can expect, at least in the case of metaheuristics, that the algorithm behaves inefficiently. Fortunately, the input size of all the analyzed binary-coded algorithms grows linearly, for example, in the case of continuous optimization. We have a population of quantum registers given by $popsize \cdot n \cdot nvar$ where *popsize* is the population size handled by the algorithm, n is the number of qubits that requires each decision variable to be coded, and *nvar* is the number of variables to be optimized. Almost all the viable analyzed algorithms need to maintain a population of quantum chromosomes. The exception is the QIAEA that uses only one quantum chromosome, hence the size complexity of the QIAEA is also smaller than the other ones.

Regarding the **lenght** of the quantum circuit of the translated quantum-inspired metaheuristics. In [48] was reported that the QIAEA had a lower median than the other QIEAs requiring 11 generations to achieve a precision constraint in the objective value of 10^{-3}, whereas the AQGA by [67]

TABLE 5. Mean and standard deviation of the GQA, AQGA and QIAEA. Best results in bold [48].

	GQA		AQGA		QIAEA	
Function	Mean / SD	Mean time (s)	Mean / SD	Mean time (s)	Mean / SD	Mean time (s)
1	1.44E-03 / 4.07E-03	25.93	3.33E-04 / 1.04E-03	26.78	2.50e-06 /SD 2.14e-06	3.92
2	5.65E+00/ 2.05E+01	27.55	1.28E+00 / 6.55E+00	27.15	**4.17e-04** / 5.11e-04	3.62
3	4.43E-01 / 5.54E-01	27.75	3.41E-01 / 6.61E-01	26.85	**3.81e-04** / 4.34e-04	4.05
4	-9.52E-01 / 2.80E-02	27.53	-9.57E-01 / 2.93E-02	26.47	**-9.70e-01** / 3.22e-02	3.45
5	1.94E-01 / 3.780E-01	27.36	1.00E-01 / 2.77E-01	26.91	**1.96e-05** / 2.27e-05	6.11
6	3.94E-03 / 5.78E-03	26.02	1.61e-03 / 4.06E-03	26.85	**2.96e-04** / 1.03e-03	3.92
7	-1.86E+02 / 4.54E-01	26.85	-1.85E+02 / 7.85E+00	26.35	**-1.87e+02** / 2.03e-03	6.49
8	9.48E-01 / 5.074E-02	25.93	**9.38E-01** / 4.86E-02	26.81	9.42e-01 / 5.01e-02	6.36
9	4.68E-02 / 6.90E-02	27.15	**5.73E-03** / 1.16E-02	27.13	2.48e-02 / 6.48e-02	4.13
10	-1.80E+00 / 4.58E-03	26.57	-1.80E+00 / 4.57E-03	26.66	**-1.80e+00** / 1.78e-06	4.00
11	**-1.02E+00** / 1.17E-02	26.11	-1.02E+00 / 1.26E-02	26.59	-1.03e+00 / 1.22e-02	3.41
12	-1.91E+01 / 1.92E-02	25.85	-1.92E+01 / 2.22E-02	26.58	**-1.92e+01** / 1.25e-05	4.03
13	2.35E+00 / 1.26E+00	26.81	2.24E+00 / 1.21E+00	26.91	**1.92e+00** / 8.27e-01	4.06
14	-2.06e+00 / 7.11e-04	26.24	-2.06e+00 / 4.95e-04	26.40	**-2.06e+00** / 1.84e-07	4.10
15	5.39e-03 / 4.45e-03	26.16	6.07e-03 / 3.63e-03	26.11	**3.52e-03** / 3.75e-03	3.95
Average	-	26.65	-	26.70	-	4.37

required 18 generations and the QGA 24 generations when optimizing the benchmark function shown in Table 4. Based on the mentioned values, one could expect that the QIAEA be $\approx$ 64% less complex than the AQGA, and $\approx$ 118% less complex than the QGA.

The RQIEA was tested using four benchmark functions [32]; two are unimodal functions (Sphere and Schwefel 2.21), and the other is multimodal (Griewank and Ackley 1). The results were compared against the stochastic genetic algorithm (StGA) [66], fast evolutionary programming (FEP) [73], fast evolutionary strategy (FES) [74], and particle swarm optimization (PSO) [3]. The reported results demonstrated that the RQIEA outperformed the FEP, FES, PSO with fewer evaluations, and the StGA with the same number of evaluated functions numerically.

Regarding the quantum-inspired version of artificial bee colonies, in [5], a metaheuristic named quantum-inspired artificial bee colony (QABC) was proposed. Similarly to the original proposal, it was tested for solving benchmark functions; particularly, the algorithm was tested with Sphere, Rosenbrock, and the Grienwank function. According to the reported results, optimizing the mentioned functions, the QABC outperformed the quantum swarm evolutionary algorithm and conventional evolutionary algorithms.

Other work that reported a common test for several algorithms is the QSE [54]. Here the QSE was compared against the GQA [22], QEA [23], QEA with probabilistic termination conditions [24], two phases TPQEA [24], advanced quantum genetic algorithm (ANQGA) [77], random search and greedy selection based GQA (RSGS_GQA) [53], quantum-inspired evolutionary algorithm based on p-system (QEPS) [76], adaptive quantum-inspired differential evolution algorithm (AQDE) [28], quantum-inspired cuckoo algorithm (QICSA) [38], quantum-inspired harmony search algorithm (QIHSA) [39], quantum-inspired DE and PSO algorithm (QDEPSO) [79], quantum-inspired Swarm Evolution (QSwEv) [68], discrete binary differential algorithm [28], binary artificial bee colony optimization and binary PSO (BABCNBPSO) [26], different binary PSO variants (DBPSO) [4]. In [54], the QSE was tested using the 0-1 knapsack problem, and it was reported that the QSE has better performance than the GQA, QEA, QEA with different termination conditions, TPQEA, ANQEA, RSGS_GQA, QEPS, AQDE, QSwEv, DBPSO, and BABCNBPSO. There was not reported a comparative of convergence time of each algorithm. For the possibility of implementing these quantum-inspired metaheuristics, all of them use several quantum registers to generate the classical population, which at present is not suitable since the number of qubits that the quantum computers using the circuit model are very limited.

TABLE 6. Hit accuracy of 10^{-3}. Population size 40, 50 generations, 40 runs. Best results in bold [48].

	QGA	AQGA	QIAEA
Function	Accuracy	Accuracy	Accuracy
1	70.00	92.5	**100.00**
2	20.00	10.00	**92.50**
3	57.50	60.00	**90.00**
4	25.00	25.00	**52.50**
5	35.00	55.00	**100.00**
6	57.5	80.00	**92.50**
7	30.00	17.50	**52.50**
8	52.00	**60.00**	57.5
9	22.50	**52.50**	40.00
10	97.50	97.50	**100.00**
11	52.50	67.50	**82.50**
12	50.00	40.00	**100.00**
13	20.00	10.00	35.00
14	57.50	77.00	**100.00**
15	35.00	25.00	**52.50**

In Tables 6 and 7, we provide a comparison of the metrics performance and execution time for the algorithms GQA, AQGA, and QIAEA. Table 6 shows the mean value, standard deviation value, and execution time of each algorithm when optimizing 15 benchmark functions, see Table 5. The values were calculated using 50 different experiments for each function. In order to calculate statistical values, each algorithm was executed 40 times for each function. The QIAEA used only one quantum chromosome (one Queen) to create a classical population. The algorithms GQA, and AQGA, used a population of 40 quantum chromosomes to create a classical

TABLE 7. Most difficult functions of the set. Best results in bold [48].

Algorithm	Function	Pop size	Generations	Accuracy	Mean / SD	Mean Time (s)
5*QGA	4	100	250	42.50	-9.6334e-01 / 3.1917e-02	342.58
	7	100	200	52.5	-1.8657e+02 / 2.8310e-01	290.21
	8	200	300	85.0	9.1500e-01 / 3.6162e-02	920.53
	9	100	200	67.5	1.2001e-02 / 2.3993e-02	284.02
	13	100	100	37.5	1.6637 / 6.9805e-01	123.399
5*AQGA	4	100	250	72.5	-9.8843e-01 / 2.3805e-02	345.93
	7	100	200	80	-1.8672e+02 / 3.6178e-02	281.04
	8	100	300	**100**	9.0000e-01 / 6.8033e-07	942.75
	9	100	200	100	4.0852e-04 / 3.3519e-04	271.41
	13	100	100	40	1.6155 / 5.9770e-01	144.52
5*QIAEA	4	100	250	**100**	-9.9999e-01 / 7.2964e-06	51.94
	7	100	200	**100**	-1.8673e+02 / 1.5695e-04	63.87
	8	200	300	92.5	9.0750e-01 / 2.6674e-02	125.86
	9	100	200	**100**	1.8057e-04 / 2.6518e-04	65.63
	13	100	100	**70**	1.5217 / 1.4454	25.45

population. All the algorithms used 40 classical individuals and run 50 iterations. In Table 6, the best results are indicated in bold; the standard deviation value was the decisive factor for those tests with the same mean value. In general, the QIAEA performed better and has shorter running times. Table 7 shows the accuracy of each algorithm for the same set of experiments. The QIAEA had the best performance in terms of precision and accuracy; in 10 functions reached the smallest mean, and in 8 functions achieved the lowest standard deviation. The second place was the AQGA, and the worst results were obtained with the GQA. In the same work, the experiments that reached an accuracy lower than 60% in the QIAEA was chosen to perform more comparative tests, in particular, the function 4, 7, 8, 9 and 13 of Table 4 were selected. The idea was to increase the population number and iteration number until the QIAEA reached an accuracy closed to 100% ;the results are shown in Table 7. This table shows that the QIAEA needed fewer generations than the other ones to achieve the best results. Hence, we could expect that this algorithm could have less complexity value since it is expected to have less depth.

IX. CONCLUSION

In mathematical optimization and computer science, a metaheuristic is a procedure that allows finding good solutions to a difficult optimization problem. A characteristic of metaheuristics is that they make few assumptions about the optimization problem, and they are useful for a variety of problems. Metaheuristics are more abstract than simple heuristics and can use low-level heuristics or search algorithms. Compared to classical optimization algorithms and iterative search methods, a metaheuristic does not guarantee to find the optimal global solution of a problem. Since 1980 when the field of metaheuristics takes off [65], and from 1996, when the concepts and principles of quantum mechanics were applied to make metaheuristics more efficient, many papers have also been published. All the quantum-based metaheuristic proposals were tested in classical computers; in some cases, the authors provided only performance results leaving aside execution times.

Many algorithms were developed long before the classical computer exist; similarly, quantum algorithms existed before the first quantum computer run; they must run on a realistic model of computation such as the circuit model. Quantum-inspired algorithms are in the middle of classical and quantum algorithms. These algorithms follow a finite sequence of instructions emulating some quantum operations such as gates, superposition, and measurement. Therefore, they might be or not distant from the circuit model. As more closed to the circuit model, more manageable is the translation.

It is worth to mention that implementing a metaheuristic in a real quantum computing based on the circuit model requires the evaluation of the fitness function, which means to measure the quantum register that will collapse the quantum register to a value. Hence, to implement a metaheuristic in a quantum computer based on the circuit model requires a hybrid programming strategy that combines a quantum computer that handles the quantum chromosome as well as the quantum operators that modify the register, and a classical computer that evaluate the fitness function and manipulate the flow of the algorithm. Additionally, the classical computer at each generation initialize the quantum register setting it according to the algorithm that we are handling, for example, it can be initialized with the state of the best classical chromosome in the evolution, returning with this action to the quantum state.

In this review, the first point was to analyze the quantum chromosome encoding. All the proposals based on qubits will facilitate the work, proposal base on the quantum angle, such as in the QSwE, also could work although additional steps will be needed to interpret quantum angles as qubits, and all the proposals based on real coding will need to much work to rethink the formulation in qubits. The above, along with the metrics width, size, and depth, we conclude that the quantum metaheuristic must have the less number of quantum chromosomes as be possible to be able to run in an up to date quantum computer such as the IBM Q; the quantum inspired acromyrmex evolutionary algorithm (QIAEA) fulfill this requirement.

The second point was to analyze which metaheuristics have reported the best results. Metrics such as performance, execution time, number of comparisons achieved versus other works were the factor to decide for the real quantum-inspired evolutionary algorithm (RQIEA) and the QIEA for solving continuous optimization problems; and the quantum social

evolution (QSE) for combinatorial optimization problems. Concerning this point, as it was mentioned in the analysis of the methods, there are many considerations to observe, so it is advisable to make more comparative tests including a diversity of quantum metaheuristics, using the same benchmark functions and combinatorial problems to record performance, and execution times or the maximal number of fitness function evaluations. Only one of the reviewed papers contained information about the execution time, but neither one contained the maximal number of fitness function evaluations that would have been very useful to make the fairest decision.

REFERENCES

[1] A. V. A. da Cruz, M. M. B. R. Vellasco, and M. A. C. Pacheco, "Quantum-inspired evolutionary algorithm for numerical optimization," in *Proc. IEEE Int. Conf. Evol. Comput.*, Jul. 2006, pp. 2630–2637.

[2] F. S. Alfares and I. I. Esat, "Real-coded quantum inspired evolution algorithm applied to engineering optimization problems," in *Proc. 2nd Int. Symp. Leveraging Appl. Formal Methods, Verification Validation (ISOLA)*, Nov. 2006, pp. 169–176.

[3] P. J. Angeline, "Evolutionary optimization versus particle swarm optimization: Philosophy and performance differences," in *Evolutionary Programming VII*, V. W. Porto, N. Saravanan, D. Waagen, and A. E. Eiben, Eds. Berlin, Germany: Springer, 1998, pp. 601–610.

[4] J. C. Bansal and K. Deep, "A modified binary particle swarm optimization for knapsack problems," *Appl. Math. Comput.*, vol. 218, no. 22, pp. 11042–11061, Jul. 2012.

[5] A. Bouaziz, A. Draa, and S. Chikhi, "A quantum-inspired artificial bee colony algorithm for numerical optimisation," in *Proc. 11th Int. Symp. Program. Syst. (ISPS)*, Apr. 2013, pp. 81–88.

[6] T. Chakraborti, A. Chatterjee, A. Halder, and A. Konar, "Automated emotion recognition employing a novel modified binary quantum-behaved gravitational search algorithm with differential mutation," *Expert Syst.*, vol. 32, no. 4, pp. 522–530, 2015.

[7] J. Chaturvedi, "Article: Adaptive quantum inspired genetic algorithm for combinatorial optimization problems," *Int. J. Comput. Appl.*, vol. 107, no. 4, pp. 34–42, Dec. 2014.

[8] M. Clerc, "The swarm and the queen: Towards a deterministic and adaptive particle swarm optimization," in *Proc. Congr. Evol. Comput. (CEC)*, vol. 3, Jul. 1999, pp. 1951–1957.

[9] M. Clerc and J. Kennedy, "The particle swarm—Explosion, stability, and convergence in a multidimensional complex space," *IEEE Trans. Evol. Comput.*, vol. 6, no. 1, pp. 58–73, Feb. 2002.

[10] S. A. Cook, "The complexity of theorem-proving procedures," in *Proc. 3rd Annu. ACM Symp. Theory Comput. (STOC)*, New York, NY, USA, 1971, pp. 151–158.

[11] D-Wave Systems. (2019). *Dwave Systems*. Accessed: Sep. 15, 2019. [Online]. Available: https://www.dwavesys.com/home

[12] L. R. da Silveira, R. Tanscheit, and M. M. Vellasco, "Quantum inspired evolutionary algorithm for ordering problems," *Expert Syst. Appl.*, vol. 67, pp. 71–83, Jan. 2017.

[13] K. Deb, *Multi-Objective Optimization Using Evolutionary Algorithms*. Chichester, U.K.: Wiley, 2001.

[14] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Trans. Evol. Comput.*, vol. 6, no. 2, pp. 182–197, Apr. 2002.

[15] M. Dorigo, V. Maniezzo, and A. Colorni, "Ant system: Optimization by a colony of cooperating agents," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 26, no. 1, pp. 29–41, Feb. 1996.

[16] K.-L. Du and M. Swamy, *Search and Optimization by Metaheuristics*. Cambridge, MA, USA: Birkhaüser, 2016.

[17] F. Fabris and R. A. Krohling, "A co-evolutionary differential evolution algorithm for solving min–max optimization problems implemented on GPU using C-CUDA," *Expert Syst. Appl.*, vol. 39, no. 12, pp. 10324–10333, 2012.

[18] D. B. Fogel, "An introduction to evolutionary computation," in *Evolutionary Computation: The Fossil Record*. Hoboken, NJ, USA: Wiley, 1998, p. 656.

[19] W. Gao, S. Liu, and L. Huang, "A global best artificial bee colony algorithm for global optimization," *J. Comput. Appl. Mathematics*, vol. 236, no. 11, pp. 2741–2753, 2012.

[20] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman, 1979.

[21] Google LLC. (2019). *Quantum, Google AI*. Accessed: Sep. 15, 2019. [Online]. Available: https://ai.google/research/teams/applied-science/quantum-ai/

[22] K.-H. Han and J.-H. Kim, "Genetic quantum algorithm and its application to combinatorial optimization problem," in *Proc. Congr. Evol. Comput. (CEC)*, vol. 2, Jul. 2000, pp. 1354–1360.

[23] K.-H. Han and J.-H. Kim, "Quantum-inspired evolutionary algorithm for a class of combinatorial optimization," *IEEE Trans. Evol. Comput.*, vol. 6, no. 6, pp. 580–593, Dec. 2002.

[24] K.-H. Han and J.-H. Kim, "Quantum-inspired evolutionary algorithms with a new termination criterion, H_ϵ gate, and two-phase scheme," *IEEE Trans. Evol. Comput.*, vol. 8, no. 2, pp. 156–169, Apr. 2004.

[25] K.-H. Han, K.-H. Park, C.-H. Lee, and J.-H. Kim, "Parallel quantum-inspired genetic algorithm for combinatorial optimization problem," in *Proc. Congr. Evol. Comput.*, vol. 2, May 2001, pp. 1422–1429.

[26] Y. He, H. Xie, T.-L. Wong, and X. Wang, "A novel binary artificial bee colony algorithm for the set-union knapsack problem," *Future Gener. Comput. Syst.*, vol. 78, pp. 77–86, Jan. 2018.

[27] V. Hosseinnezhad, M. Rafiee, M. Ahmadian, and M. T. Ameli, "Species-based quantum particle swarm optimization for economic load dispatch," *Int. J. Elect. Power Energy Syst.*, vol. 63, pp. 311–322, Dec. 2014.

[28] A. R. Hota and A. Pat, "An adaptive quantum-inspired differential evolution algorithm for 0–1 knapsack problem," in *Proc. 2nd World Congr. Nature Biol. Inspired Comput. (NaBIC)*, Dec. 2010, pp. 703–708.

[29] Y. Huang, Y. Wang, W. Zhou, Z. Yu, and C. Zhou, "A fuzzy neural network system based on generalized class cover and particle swarm optimization," in *Advances in Intelligent Computing*, D.-S. Huang, X.-P. Zhang, and G.-B. Huang, Eds. Berlin, Germany: 2005, pp. 119–128.

[30] IBM. (2019). *IBM Q, Quantum Computing*. Accessed: Jul. 2, 2019. [Online]. Available: https://www.research.ibm.com/ibm-q/

[31] Intel Corporation. (2019). *2018 CES: Intel Advances Quantum and Neuromorphic Computing Research*. Accessed: Sep. 15, 2019. [Online]. Available: https://newsroom.intel.com/news/intel-advances-quantum-neuromorphic-computing-research/

[32] M. Jamil and X. Yang, "A literature survey of benchmark functions for global optimization problems," *Int. J. Math. Model. Numer. Optim.*, Vol. 4, no. 2, pp. 150–194, 2013.

[33] C. Jin and S.-W. Jin, "Automatic image annotation using feature selection based on improving quantum particle swarm optimization," *Signal Process.*, vol. 109, pp. 172–181, Apr. 2015.

[34] D. Karaboga, "An idea based on honey bee swarm for numerical optimization," Erciyes Univ., Kayseri, Turkey, Tech. Rep. TR06, Oct. 2005.

[35] S. Karmakar, A. Dey, and I. Saha, "Use of quantum-inspired metaheuristics during last two decades," in *Proc. 7th Int. Conf. Commun. Syst. Netw. Technol. (CSNT)*, 2017, pp. 272–278.

[36] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proc. Int. Conf. Neural Netw. (ICNN)*, vol. 4, Nov. 1995, pp. 1942–1948.

[37] Y. Kim, J.-H. Kim, and K.-H. Han, "Quantum-inspired multiobjective evolutionary algorithm for multiobjective 0/1 knapsack problems," in *Proc. IEEE Int. Conf. Evol. Comput.*, Jul. 2006, pp. 2601–2606.

[38] A. Layeb, "A novel quantum inspired cuckoo search for knapsack problems," *Int. J. Bio-Inspired Comput.*, vol. 3, no. 5, pp. 297–305, Sep. 2011.

[39] A. Layeb, "A hybrid quantum inspired harmony search algorithm for 0–1 optimization problems," *J. Comput. Appl. Math.*, vol. 253, pp. 14–25, Dec. 2013.

[40] B. Li and L. Wang, "A hybrid quantum-inspired genetic algorithm for multiobjective flow shop scheduling," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 37, no. 3, pp. 576–591, Jun. 2007.

[41] X. L. Li, Z. J. Shao, and J. X. Qian, "An optimizing method based on autonomous animats: Fish-swarm algorithm," *Syst. Eng.-Theory Pract.*, vol. 22, no. 11, p. 32, 2002.

[42] Y. Li, H. Shi, L. Jiao, and R. Liu, "Quantum evolutionary clustering algorithm based on watershed applied to SAR image segmentation," *Neurocomputing*, vol. 87, no. 10, pp. 90–98, Jun. 2012.

[43] C. McGeoch and C. Wang, "Experimental evaluation of an adiabiatic quantum system for combinatorial optimization," in *Proc. ACM Int. Conf. Comput. Frontiers (CF)*, 2013.

[44] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*. Berlin, Germany: Springer, 1996.
[45] M. S. Moghadam, H. Nezamabadi-Pour, and M. M. Farsangi, "A quantum behaved gravitational search algorithm," *Intell. Inf. Manage.*, vol. 2012, no. 4, pp. 390–395, 2012.
[46] O. Montiel, O. Castillo, P. Melin, A. R. Díaz, and R. Sepúlveda, "Human evolutionary model: A new approach to optimization," *Inf. Sci.*, vol. 177, no. 10, pp. 2075–2098, 2007.
[47] O. Montiel, O. Castillo, P. Melin, and R. Sepulveda, "Mediative fuzzy logic: A new approach for contradictory knowledge management," *Soft Comput.*, vol. 12, no. 3, pp. 251–256, Feb. 2008.
[48] O. Montiel, Y. Rubio, C. Olvera, and A. Rivera, "Quantum-inspired acromyrmex evolutionary algorithm," *Sci. Rep.*, vol. 9, Aug. 2019, Art. no. 12181.
[49] P. Moore and G. K. Venayagamoorthy, "Evolving combinational logic circuits using a hybrid quantum evolution and particle swarm inspired algorithm," in *Proc. NASA/DoD Conf. Evolvable Hardw. (EH)*, Jun. 2005, pp. 97–102.
[50] A. Narayanan, "Quantum computing for beginners," in *Proc. Congr. Evol. Comput. (CEC)*, vol. 3, Jul. 1999, pp. 2231–2238.
[51] A. Narayanan and M. Moore, "Quantum-inspired genetic algorithms," in *Proc. IEEE Int. Conf. Evol. Comput.*, May 1996, pp. 61–66.
[52] R. Pavithr and Gursaran, "Quantum inspired social evolution (QSE) algorithm for 0–1 knapsack problem," *Swarm Evol. Comput.*, vol. 29, pp. 33–46, Aug. 2016.
[53] R. S. Pavithr and Gursaran, "A random search and greedy selection based genetic quantum algorithm for combinatorial optimization," in *Proc. IEEE Congr. Evol. Comput.*, Jun. 2013, pp. 2422–2427.
[54] R. S. Pavithr and Gursaran, "Social evolution: An evolutionary algorithm inspired by human interactions," in *Proc. 2nd Int. Conf. Soft Comput. Problem Solving (SocProS)*, B. V. Babu, A. Nagar, K. Deep, M. Pant, J. C. Bansal, K. Ray, and U. Gupta, Eds. New Delhi, India: Springer, 2014, pp. 1537–1544.
[55] QuTech. (2019). *Research and Development in Quantum Technology*. Accessed: Jul. 2, 2019. [Online]. Available: https://qutech.nl
[56] E. Rashedi, H. Nezamabadi-pour, and S. Saryazdi, "GSA: A gravitational search algorithm," *Inf. Sci.*, vol. 179, no. 13, pp. 2232–2248, 2009.
[57] E. Rashedi, H. Nezamabadi-Pour, and S. Saryazdi, "BGSA: Binary gravitational search algorithm," *Natural Comput.*, vol. 9, no. 3, pp. 727–745, Sep. 2010.
[58] E. Rashedi, E. Rashedi, and H. Nezamabadi-Pour, "A comprehensive survey on gravitational search algorithm," *Swarm Evol. Comput.*, vol. 41, pp. 141–158, Jun. 2018.
[59] Rigetti. (2019). *Rigetti Computing Quantum Cloud Services*. Accessed: Sep. 15, 2019. [Online]. Available: https://www.rigetti.com/qcs
[60] N. M. Sabri, M. Puteh, and M. R. Mahmood, "A review of gravitational search algorithm," *Int. J. Adv. Soft Comput. Appl.*, vol. 5, no. 3, pp. 1–39, 2013.
[61] J. Senthilnath, S. Omkar, and V. Mani, "Clustering using firefly algorithm: Performance study," *Swarm Evol. Comput.*, vol. 1, no. 3, pp. 164–171, 2011.
[62] Y. Shi and R. Eberhart, "A modified particle swarm optimizer," in *Proc. IEEE Int. Conf. Evol. Comput. IEEE World Congr. Comput. Intell.*, May 1998, pp. 69–73.
[63] K. Socha and M. Dorigo, "Ant colony optimization for continuous domains," *Eur. J. Oper. Res.*, vol. 185, no. 3, pp. 1155–1173, 2008.
[64] M. M. Soliman, A. E. Hassanien, and H. M. Onsi, "An adaptive watermarking approach based on weighted quantum particle swarm optimization," *Neural Comput. Appl.*, vol. 27, no. 2, pp. 469–481, 2016.
[65] K. Sorensen, M. Sevaux, and F. Glover, "A history of metaheuristics," in *Handbook of Heuristics*, R. Martí, P. Panos, and M. Resende, Eds. Cham, Switzerland: Springer, 2018, pp. 1–18.
[66] Z. Tu and Y. Lu, "A robust stochastic genetic algorithm (STGA) for global numerical optimization," *IEEE Trans. Evol. Comput.*, vol. 8, no. 5, pp. 456–470, Oct. 2004.
[67] H. Wang, J. Liu, J. Zhi, and C. Fu, "The improvement of quantum genetic algorithm and its application on function optimization," *Math. Problems Eng.*, vol. 2013, 2013, Art. no. 730749.
[68] Y. Wang, X.-Y. Feng, Y.-X. Huang, D.-B. Pu, W.-G. Zhou, Y.-C. Liang, and C.-G. Zhou, "A novel quantum swarm evolutionary algorithm and its applications," *Neurocomputing*, vol. 70, no. 4, pp. 633–640, 2007.
[69] C. P. Williams, *Explorations in Quantum Computing*, 2nd ed. New York, NY, USA: Springer-Verlag, 2011.
[70] D. P. Williamson and D. B. Shmoys, *The Design of Approximation Algorithms*, 1st ed. New York, NY, USA: Cambridge Univ. Press, 2011.
[71] S. Yang, M. Wang, and L. Jiao, "A quantum particle swarm optimization," in *Proc. Congr. Evol. Comput.*, vol. 1, Jun. 2004, pp. 320–324.
[72] X. Yang, *Nature-Inspired Metaheuristic Algorithms*, 2nd ed. Frome, U.K.: Luniver Press, 2010.
[73] X. Yao, Y. Liu, and G. Lin, "Evolutionary programming made faster," *IEEE Trans. Evol. Comput.*, vol. 3, no. 2, pp. 82–102, Jul. 1999.
[74] X. Yao and Y. Liu, "Fast evolution strategies," in *Evolutionary Programming VI*, P. J. Angeline, R. G. Reynolds, J. R. McDonnell, and R. Eberhart, Eds. Berlin, Germany: Springer, 1997, pp. 149–161.
[75] G. Zhang, "Quantum-inspired evolutionary algorithms: A survey and empirical study," *J. Heuristics*, vol. 17, no. 3, pp. 303–351, Jun. 2011.
[76] G. Zhang, M. Gheorghe, and C. Wu, "A quantum-inspired evolutionary algorithm based on P systems for knapsack problem," *Fundam. Inform.*, vol. 87, no. 1, pp. 93–116, 2008.
[77] G. Zhang, N. Li, W. dong Jin, and L. zhao Hu, "Novel quantum genetic algorithm and its applications," *Frontiers Electr. Electron. Eng. China*, vol. 1, no. 1, pp. 31–36, Jan. 2006.
[78] K. Zhu and M. Jiang, "Quantum artificial fish swarm algorithm," in *Proc. 8th World Congr. Intell. Control Automat.*, 2010, pp. 1–5.
[79] D. Zouache and A. Moussaoui, "Quantum-inspired differential evolution with particle swarm optimization for knapsack problem," *J. Inf. Sci. Eng.*, vol. 31, pp. 1779–1795, Sep. 2015.
[80] D. Zouache, F. Nouioua, and A. Moussaoui, "Quantum-inspired firefly algorithm with particle swarm optimization for discrete optimization problems," *Soft Comput.*, vol. 20, no. 7, pp. 2781–2799, Jul. 2016.

OSCAR H. MONTIEL ROSS (Senior Member, IEEE) received the M.Sc. degree in digital systems from the Instituto Politécnico Nacional (IPN), in 1999, the M.Sc. degree in computer science from the Tijuana Institute of Technology, Tijuana, México, in 2000, and the D.Sc. degree in computer science from the Universidad Autónoma of Baja California, Tijuana, in 2006. He is currently a Researcher with the Centro de Investigación y Desarrollo de Tecnología Digital (CITEDI), IPN. He has published articles about quantum computing, evolutionary computation, mobile robotics, Mediative fuzzy logic, ant colonies, type-2 fuzzy systems, and embedded systems. He is a member of the International Association of Engineers (IANG), and the Mexican Science Foundation CONACYT (Consejo Nacional de Ciencia y Tecnología). He received the Research Award 2016 from IPN. He is a Co-Founder and an Active Member of the HAFSA (Hispanic American Fuzzy Systems Association), and the Mexican Chapter of the Computational Intelligence Society (IEEE).