ISSN 2007-9737

Comparison of Performance of Amazon Braket Using a Quantum Genetic Algorithm

Sandra S. Rosales[1], Oscar Montiel[1], Ulises Orozco-Rosas[2], Juan J. Tapia[1], Oscar Castillo[3,*]

[1] Instituto Politécnico Nacional,
Centro de Investigación y Desarrollo de Tecnología Digital,
Mexico

[2] El Centro de Enseñanza Técnica y Superior,
Mexico

[3] Instituto Tecnológico de Tijuana,
Mexico

{oross, jtapiaa}@ipn.mx, srosales@citedi.mx,
ulises.orozco@cetys.mx, ocastillo@tectijuana.mx

Abstract. This study evaluates the performance of a Quantum Genetic Algorithm (QGA) executed on quantum devices. The algorithm was implemented in MATLAB and Python and deployed on the Amazon Web Services (AWS) platform. The QGA was utilized to optimize a set of continuous single-variable functions. The implementation employed the Hadamard quantum gate to initialize the population and the R_y rotation gate for mutation and crossover. The findings revealed significant differences in execution time and costs were observed between the two implementations, undescoring the performance of quantum devices available on AWS. The results demonstrate that the QGA can achieve optimal solutions in a few generations, suggesting its potential for efficiently solving complex problems. However, the costs and availability of quantum devices remain restrictive. This work exemplifies the potential of leveraging AWS cloud-based quantum computing platforms for the research and development of quantum algorithms.

Keywords. Quantum computing, quantum genetic algorithms, mathematical optimization, quantum device on AWS, quantum metaheuristic algorithm.

1 Introduction

Quantum computing has experienced significant growth in both the development of devices and the application of quantum algorithms across various fields [8]. One prominent area is optimization, which can be approached using different methods, including evolutionary algorithms [25].

In the context of our work, optimization involves the process of maximizing or minimizing a continuous single-variable function. The concept of quantum evolutionary algorithms was first introduced by Narayanan and Moore in 1996 [19]. Their work laid the groundwork by applying the principles of quantum mechanics to the evolution of a quantum particle over time.

This idea has been enriched over the years, and various applications in quantum simulators have existed. For example, some studies such as [3, 10] applied this approach to combinatorial optimization, while [17] used it for optimization functions.

Different proposals for applying this method are generally discussed in works like [14, 18].

Algorithm 1 Quantum genetic algorithm (QGA)

Require: g, s_p ▷ g: amount of generations, s_p: population size
Ensure: The best solution **b**
procedure QGA(g, s_p)
  $t \leftarrow 0$
  Set up a quantum population $Q_t(s_p)$ with quantum gate H
  Measure each individual in $Q_t(s_p)$ to create a population $P_t(s_p)$
  Save the best solution **b** in $P_t(s_p)$
  while $t <$ g **do** ▷ t: indicate the number of generation
    Update $Q_t(s_p - 1)$ applying a quantum rotation gate $U(\Delta\theta_j)$
    Measure each individual in $Q_t(s_p - 1)$ to create a population $P_t(s_p - 1)$
    Save the best solution **b** in $P_t(s_p - 1)$
    $t \leftarrow t + 1$
  end while
end procedure

Quantum computing offers significant potential for solving problems that classical computing cannot efficiently address. However, it is important to note that current quantum devices are in the NISQ (Noisy Intermediate-Scale Quantum) era, meaning that their results may include errors.

These errors can be mitigated by repeating executions, which unfortunately makes the process slow and costly. Therefore, the problems addressed must be adapted to the current limitations of these quantum devices. As technology advances, it is anticipated that the need for repeated measurements will decrease.

Amazon Braket, a service from Amazon Web Services (AWS), has emerged as a valuable tool providing access to quantum devices from various vendors. This platform enables users to work with different types of quantum devices, as discussed in Section 3. Access to these quantum devices can be achieved through Python or MATLAB. In Section 3 we review the pseudocode of a Quantum Genetic Algorithm (QGA) that was coded and executed in MATLAB and Python.

This section details how to establish a connection to AWS, enabling code execution on quantum devices. Additionally, Section 4 presents the results of implementation using both languages. The goal of this study is to compare the performance of a QGA implemented to maximize a set of continuous single-variable functions using both Python and MATLAB.

The structure of this paper is as follows: Section 2 covers the main concepts and foundational knowledge employed in this study. Section 3 explores the implementation details, algorithms, functional aspects, and provides a brief cost analysis of using AWS for quantum computing. Section 4 describes the findings from applying this methodology with both a simulator and a quantum device. The final Section 5 includes the conclusions and suggests potential avenues for future research.

2 Theoretical Framework

This section presents a review of problem optimization and fundamentals of quantum computing that will aid in understanding Quantum Genetic Algorithms (QGAs).

2.1 Optimization Problem

Optimization problems are prevalent across various fields, including engineering, logistics, finance, and artificial intelligence. These problems involve finding the optimal solution, often within complex constraints or large solution spaces [20]. Traditional optimization techniques, such as linear programming or gradient descent, are effective for well-structured problems but frequently struggle with highly nonlinear, multi-modal, or combinatorial optimization challenges [6].

These conventional methods often encounter difficulties in identifying optimal solutions within a practical time frame. This is where metaheuristic methods become essential [4]. Metaheuristic methods encompass a class of optimization algorithms designed to tackle complex and computationally demanding optimization problems. Unlike deterministic approaches, metaheuristics rely on heuristic or rule-of-thumb strategies, avoiding explicit mathematical models [11]. These algorithms explore solution spaces by adeptly

Table 1. Values and conditions for rotation $\Delta\theta_i$[1]. $\mathbf{X_i}$ depict the $i-th$ decimal value proposed. $\mathbf{B_i}$ depict the $i-th$ decimal value stored

$\mathbf{x_{ij}}$	$\mathbf{b_{ij}}$	$\mathbf{f(X_i) \geq f(B_i)}$	$\mathbf{\Delta\theta_i}$
0	0	$false$	0
0	0	$true$	0
0	1	$false$	0
0	1	$true$	-0.05π
1	0	$false$	-0.01π
1	0	$true$	0.025π
1	1	$false$	0.005π
1	1	$true$	0.025π

Algorithm 2 Update of the QGA

Require: q_i, x_i, b_i ▷
q_i quantum individual, x_i represents the best element, while b_i represents the new candidate element in binary form. ▷ $f(\mathbf{B_i})$ and $f(\mathbf{X_i})$ where $\mathbf{B_i}$ and $\mathbf{X_i}$ are decimal values
Ensure: Rate of evolution
1: **procedure** UPDATE(q_i)
2: $i \leftarrow 0$
3: **while** $i < m$ **do** ▷ m is the length of q_i
4: Calculate $\Delta\theta_i$ and $s(\alpha'_i, \beta'_i)$ with to Table 1
5: Update q_i by applying :
6: $[\alpha'_i\beta'_i]^T = U(\Delta\theta_i)[\alpha_i, \beta_i]^T$
7: $i \leftarrow i + 1$
8: **end while**
9: $q_i = q'_i$
10: **end procedure**

balancing exploration to discover new areas of the landscape and exploitation to refine promising solutions.

The strength of metaheuristics lies in their adaptability and versatility, as they can be customized for various problem types and constraints. Additionally, they excel in solving real-world challenges where exact solutions are often impractical due to computational limitations [11]. As technology advances and optimization problems become increasingly complex, the role of metaheuristics continues to expand, establishing them as indispensable tools for researchers and practitioners tackling real-world optimization issues.

Furthermore, quantum computing and recent proposals of quantum metaheuristics enhance the search process within the optimization landscape, opening new frontiers for addressing previously unsolvable problems [7].

The objective of optimization is to find the best possible solution, referred to as *feasible solutions*, which are measured using numerical functions, referred to as *objective functions* [15]. In the *feasible solution* set, the solution that yields the best objective function value is referred to as the *optimal solution* [5]. The formal definition is shown in:

Let $f : X \to \mathbb{R}$ be a function where $X \subseteq \mathbb{R}^n$ $n-$dimensional in the form $x = [x_1, \ldots, x_n]^T$ The aim is to:

$$\begin{array}{ll} f(x)_{\text{max}} & \text{or} \quad f(x)_{\text{min}}, \\ x \in X_{\text{max}} & x \in X_{\text{min}}, \end{array} \tag{1}$$

where $x_j, j = 1, \ldots, n$ is identified as a *decision variable*, X refers to the *feasible region* and f is the *objective function*.

2.2 Quantum Computing Fundamentals

In quantum computing, the basic unit of information is a quantum bit commonly called a *qubit*, which has a form $|\psi\rangle$[21]. Quantum computing mathematically represents qubits using Dirac notation, also referred to as bra-ket notation. For example, the ket $|i\rangle = \begin{bmatrix} 1 & i \end{bmatrix}^T$ where $i \in \mathbb{C}$, and the ket $|1\rangle = \begin{bmatrix} 0 & 1 \end{bmatrix}^T$. A qubit is a linear combination of $|0\rangle$ and $|1\rangle$.

Bra and ket vectors are complex vectors within the Hilbert space, existing in a dual space. The notation $\langle\psi|$ indicates a bra, which is the complex conjugate transpose of the ket vector $|\psi\rangle$. Hence, $\langle\psi| = |\psi\rangle^\dagger$, where the symbol † signifies the complex conjugate transpose operation.

In quantum computing, computational bases are fundamental for representing and manipulating information. The most common computational basis consists of the states $|0\rangle$ and $|1\rangle$, which are

Table 2. Types of quantum computers: Available quantum devices on Amazon braket. Source: The table is from the chapter [22]

Companies	Quantum device names	Quantum computing technologies
IonQ	Harmony, Aria-1	Through the utilization of precise laser pulses, this system monitors ions that are confined in space to execute quantum gate operations and measurements. The systems are equipped with 11 and 25 qubits respectively.
Oxford Quantum Circuits (OQC)	Lucy, Aspen, M-3	Its technology utilizes superconducting quantum processors, having 8 qubits. In contrast, Rigetti is composed of two chips with a total of 79 qubits.
QuEra Computing	Aquila	The principle behind this technology is to use lasers to arrange and excite neutral atoms into highly energetic states. Their quantum computer consists of 256 qubits operating in analog mode. Analog Hamiltonian Simulation is his paradigm.

analogous to the binary bits "0" and "1" in classical computing. These states serve as the foundation for encoding and processing quantum information. The *superposition principle* in quantum mechanics permits a qubit to be in a linear combination of the two fundamental states, $|0\rangle$ and $|1\rangle$. This relationship is illustrated in Eq. 2:

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle \text{ with } \alpha, \beta \in \mathbb{C}. \tag{2}$$

In Eq. 2, α and β are complex numbers representing the probability amplitudes. When measuring a qubit, the likelihood of finding the qubit in state $|0\rangle$ or $|1\rangle$ is given by the square of the absolute value of α or β, respectively.

The normalization condition requires that the sum of these squares equals one: $|\alpha|^2 + |\beta|^2 = 1$. Classical computers use registers to store bits, likewise, quantum computers register and store qubits. Despite this, a quantum qubit register and a classical bit register differ in important ways.

Quantum bits, unlike classical bits, are in a superposition of states. This enables them to represent both "0" and "1" simultaneously, enabling quantum computers to handle multiple possibilities simultaneously. Furthermore, qubits are capable of becoming *entangled*, enabling a profound link between their states, despite being physically separated, which facilitates more efficient calculations. Eq. 3 represents a quantum register of size n mathematically. A tensorial product of two states (vectors) is represented by the symbol $\otimes$:

$$\begin{aligned} |\Psi\rangle &= |\psi_1\rangle \otimes ... \otimes |\psi_n\rangle = |101\rangle = |1\rangle \otimes |0\rangle \otimes |1\rangle\,, \\ &= \begin{pmatrix}0\\1\end{pmatrix} \otimes \begin{pmatrix}1\\0\end{pmatrix} \otimes \begin{pmatrix}0\\1\end{pmatrix} = \begin{pmatrix}0\\0\\0\\0\\0\\1\\0\\0\end{pmatrix}. \end{aligned} \tag{3}$$

Quantum gates are fundamental in quantum computing. They function similarly to classical logic gates in traditional computing, enabling control over the state of one or more qubits. Thanks to quantum gates, quantum computers can carry out specific tasks much faster than classical computers, as quantum states can exist in superposition and be entangled [27].

Operations can be performed on individual qubits or on groups of qubits, known as quantum registers. Some one-qubit quantum gates originate from the Pauli set. For instance, the $X = \left[\begin{smallmatrix}0 & 1\\ 1 & 0\end{smallmatrix}\right]$ gate, which induces rotations around the X-axis, and the $Y = \left[\begin{smallmatrix}0 & -i\\ i & 0\end{smallmatrix}\right]$ gate, which causes rotations around the Y-axis.

Another significant one-qubit gate frequently used to bring a qubit into a superposition state

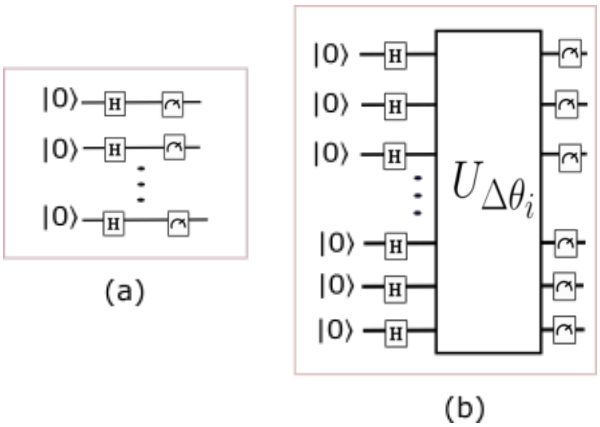


Fig. 1. The functions of the QGA are illustrated in the following diagram. (a) The quantum circuit for initializing the first population. (b) The quantum circuit for mutation and crossover operators. Figures taken from [22]

is the Hadamard gate, $H = \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}$. This superposition creates equal probabilities for the qubit to be in the $|0\rangle$ or $|1\rangle$ states, with the $|1\rangle$ state having a relative phase difference of π radians.

A one-qubit quantum gate acts on a quantum state $|\psi\rangle$ as $U|\psi\rangle$, where U is a quantum operator such as the quantum gate H. For instance, applying the Hadamard gate H to the quantum state $|0\rangle$ yields $H|0\rangle = \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}\begin{bmatrix}1\\ 0\end{bmatrix} = \frac{1}{\sqrt{2}}\begin{bmatrix}1\\ 1\end{bmatrix}$, which simplifies to $H|0\rangle = \frac{|0\rangle+|1\rangle}{\sqrt{2}}$. Another type of one-qubit gate is the rotation gate, which can be represented as:

$$R_x(\theta) = \begin{bmatrix}\cos\frac{\theta}{2} & -i\sin\frac{\theta}{2}\\ -i\sin\frac{\theta}{2} & \cos\frac{\theta}{2}\end{bmatrix},$$
$$R_y(\theta) = \begin{bmatrix}\cos\frac{\theta}{2} & -\sin\frac{\theta}{2}\\ \sin\frac{\theta}{2} & \cos\frac{\theta}{2}\end{bmatrix}, \quad (4)$$
$$R_z(\theta) = \begin{bmatrix}e^{-\frac{i\theta}{2}} & 0\\ 0 & e^{\frac{i\theta}{2}}\end{bmatrix}.$$

In Eq.4, $R_x(\theta)$ represents a rotation about the x-axis, $R_y(\theta)$ denotes a rotation about the y-axis, and $R_z(\theta)$ indicates a rotation around the z-axis. The variable θ stands for the angle of rotation for each respective axis. As mentioned earlier, a quantum register consists of multiple qubits grouped together as a single entity.

Similar to single quantum gates, in a quantum register, a quantum gate denoted as U_r can operate on the quantum register state $|\psi_r\rangle^n$ of n qubits in the form $U_r|\psi_r\rangle^n$. For example, the application of the $H^{\otimes 2}$ gate (which is two Hadamard gates in parallel, i.e., $H \otimes H$) on a two-qubit register $|\psi_r\rangle^2$ is represented as $H^{\otimes 2}|\psi_r\rangle^2$, which can be expressed as:

$$H^{\otimes 2}|\psi_r\rangle^2 = \frac{1}{2}\begin{bmatrix}1 & 1 & 1 & 1\\ 1 & -1 & 1 & -1\\ 1 & 1 & -1 & -1\\ 1 & -1 & -1 & 1\end{bmatrix}\begin{bmatrix}S_0\\ S_1\\ S_2\\ S_3\end{bmatrix}, \quad (5)$$
$$= \frac{1}{2}\begin{bmatrix}S'_0\\ S'_1\\ S'_2\\ S'_3\end{bmatrix}.$$

In this case, $S_0 \cdots S_3$ represent the quantum states of $|\psi_r\rangle^2$, and $S'_0 \cdots S'_3$ denote the quantum states that result from applying the quantum gate $H^{\otimes 2}$ to the state $|\psi_r\rangle^n$. In general, for a n-qubit state in the computational basis $|0\cdots 0\rangle_n$, applying the Hadamard gate to each qubit results in:

$$|\psi_r\rangle^n = \frac{1}{\sqrt{2^n}}(|0\cdots 000\rangle + |\cdots 001\rangle + \cdots + |1\cdots 111\rangle). \quad (6)$$

This operation with the Hadamard gate is crucial, as many quantum algorithms use it as an initial state. It enables all the 2^n orthogonal qubits in the basis states $|0\rangle$ and $|1\rangle$ to be placed into a superposition state with equal probabilities. In general, the procedure can be expressed as:

$$H^{\otimes n}|x\rangle^n = \frac{\sum_z (-1)^{x\odot z}|z\rangle}{\sqrt{2^n}}, \quad (7)$$

where $x \odot z$ denote the bit-by-bit inner product of x and z.

Measurement plays a fundamental role in quantum mechanics, offering insights into physical observables and their associated probabilities. It is essential to understand that measuring a quantum system disturbs it, causing an irreversible change in its state. In quantum computing, measurement is

Table 3. Cost by quantum device per quantum task and shot. Information taken from Amazon braket pricing

QPU family	Quantum task price (USD)	Per-shot price (USD)[2]
Harmony	$0.30	$0.01
Aria	$0.30	$0.03
Lucy	$0.30	$0.00035
Aspen-M3	$0.30	$0.00035

Table 4. Description of total quantum task, final shots, and final cost per experiment

Experiment (shots)	Total quantum tasks	Total shots	Final cost (USD)
MATLAB (100)	583	58,300	$ 803
Python (1,000)	534	534,000	$ 4,211
		Total cost	$ 5,014

particularly significant for retrieving the information encoded within the computational system. In the domain of quantum mechanics, a range of measurement models are utilized to illustrate the interaction between a quantum system and a measurement apparatus, as well as the acquisition of measurement outcomes [21].

A variety of measurement models are used such as the Projection Model, Expectation Value Model, State Collapse Model, Statistical Model, and Eigenvalue and Eigenspace Model.

The standard notation of measurements consists of a measurement operator represented as M_m, where m is used as an index for a possible measurement outcome. If state is $|\psi\rangle$, the probability of getting a measurement result m is as the following:

$$Pr(m) = \langle\psi| M_m^\dagger M_m |\psi\rangle . \tag{8}$$

Upon obtaining this measurement, the subsequent state will be:

$$|\psi'\rangle = \frac{M_m |\psi\rangle}{\sqrt{\langle\psi| M_m^\dagger M_m |\psi\rangle}}. \tag{9}$$

Quantum computing depends on measuring a group of qubits “in the computational basis”, which consists of the states $|0\rangle$ and $|1\rangle$. In this particular scenario, we consider the spin direction of individual qubits in the quantum memory register along the z-axis of the Bloch sphere, which traverses both the North and South poles of the sphere.

The results of these measurements indicate that each qubit is either aligned with the z-axis, with “spin-up”, or it is in opposition, with “spin-down”, which corresponds to state $|0\rangle$ and $|1\rangle$ respectively. If this measure is applied to each qubit in a quantum memory register comprising n-qubits, will be obtained one of 2^n possible bit strings configurations.

The generation of different outcomes is contingent upon the superposition of each binary string configuration present in the register immediately before measurement. To illustrate this, let us consider an n-qubit quantum memory register which is in the normalized state $\sum_{i=0}^{2^n-1} c_i |i\rangle$ (where $|i\rangle$ represents a bit-string).

There will be variability in the outcome depending on the magnitude of the amplitudes c_i and whether it is making a full measurement (measuring all the qubits) or a partial measurement (measuring only a few qubits). The result will be $|i\rangle$, with a probability of $|c_i|^2$ for each state.

2.3 Quantum Genetic Algorithm

An evolutionary algorithm represents an optimization approach inspired by biological evolution, used to identify or approximate solutions to intricate problems spanning various disciplines [16]. These algorithms commence with a population of potential solutions, subjecting each to fitness evaluation via a predefined function.

Selection for reproduction is based on fitness, leading to the generation of new solutions through crossover and mutation. This iterative process unfolds across multiple generations [11]. Our work focuses on Quantum Genetic Algorithms (QGAs) [9], which are rooted in Quantum-Inspired Genetic Algorithms (QIGAs) [19]. The two types of algorithms are grounded in quantum principles but have distinct implementation and computational frameworks. Quantum-Inspired Genetic Algorithms (QIGAs) are classical

Table 5. Test functions

Function	Global maximum $(x, f(x)),\ x \in [0, 127]$
$f_1(x) = \frac{x}{10} \cdot \sin(\frac{x}{10})$	(79.7867, 7.9167)
$f_2(x) = \frac{x}{2} + 5$	(127.0, 68.5000)
$f_3(x) = -\frac{\cos(x) \cdot x^2}{10} + 3x$	(122.5400, 1,868.9900)
$f_4(x) = \frac{(x - 135\pi)(\sin(\frac{x+135\pi}{10}))}{10}$	(15.4535, 39.2826)
$f_5(x) = \frac{-x^2}{10} + 3x$	(15.0800, 22.5000)
$f_6(x) = \begin{cases} \frac{\sin(x-20)}{x-20} & if \quad x \neq 20 \\ 0.9999 & if \quad x = 20 \end{cases}$	(19.9946, 0.9999)

Table 6. The best maximum found in two executions, one with 10 runs and the other with 40 runs, each consisting of 15 generations, was compared between *Harmony* and SV1 versus the local simulator on MATLAB

Function f_n	AWS Device MATLAB	Local Simulator MATLAB
f_1	(80.0625, **7.9136**)	(81.9910, **7.9167**)
f_2	(124.0000,67.0000)	(127.0000,**68.5000**)
f_3	(116.1875,1696.7312)	(122.6571,**1,868.9434**)
f_4	(15.4375, **39.2826**)	(18.6126, **39.2826**)
f_5	(15.0000, **22.5000**)	(15.0762, **22.5000**)
f_6	(20.0625, **0.9993**)	(19.9775, **0.9999**)

algorithms that run on conventional computers. They are inspired by quantum principles and seek to replicate quantum effects using classical computing methods [23].

In contrast, QGAs are specifically created to operate on authentic quantum computers, exploiting the complete potential of quantum characteristics. They possess the capacity for substantial acceleration in solving certain problems, however, their practical utility is currently constrained by the early stage of quantum hardware development and limited accessibility.

The specific approach of this study is implementing a Quantum Genetic Algorithm (QGA) using two programming languages that allow connection to quantum computers via AWS. In MATLAB, the *'quantum'* package developed in 2023 was used. For Python, the *Amazon Braket* library was utilized. It is important to mention that these tools also allow simulations to be run on personal computers.

As discussed in [18], we have customized QGA for execution on quantum computers. We take the proposal developed in [22] where " q_i characterizes a quantum system $|\psi_i\rangle$ that encompasses 2^m simultaneous states, as expressed in Eq. 10. Here, m represents the genetic composition of each individual, and i signifies the population size":

$$|\psi_i\rangle = \begin{bmatrix} \alpha_1 & \alpha_2 & \dots & \alpha_m \\ \beta_1 & \beta_2 & \dots & \beta_m \end{bmatrix} = q_i. \tag{10}$$

The outcome will produce a quantum population structured in the following manner:

$$\begin{bmatrix} \alpha_1 & \alpha_2 & \dots & \alpha_m \\ \beta_1 & \beta_2 & \dots & \beta_m \end{bmatrix}_{q_1}, \\ \begin{bmatrix} \alpha_1 & \alpha_2 & \dots & \alpha_m \\ \beta_1 & \beta_2 & \dots & \beta_m \end{bmatrix}_{q_2}, \\ \vdots \\ \begin{bmatrix} \alpha_1 & \alpha_2 & \dots & \alpha_m \\ \beta_1 & \beta_2 & \dots & \beta_m \end{bmatrix}_{q_i}. \tag{11}$$

As a result, in [22], Algorithm 1 and Algorithm 2 were developed. The Algorithm 1 requires two elements: the total amount to the evolution of population, i.e., generations, g, and the population size $s_p \rightarrow i$. In more detail on line 2, the generation number t is adjusted to 0.

In line 3, we applied the quantum Hadamard gate to all the qubits, as defined in Eq. 7 to produce equal chance distributions for all potential individuals. In line 4, we obtain a classical population by measuring all individuals in the current population. In line 5, we store the best solution in the current population in "**b**".

The while-loop, spanning from lines 6 to 10, runs continuously as long as the generation counter remains less than the maximum number, indicated as "g". The process involves generating a new quantum population using the best solution "**b**" in line 7, followed by an update using Algorithm 2. A new classical population is then formed by measuring the individuals obtained in line 7 and line 8. In line 9, the best solution from the entire process is saved as "**b**". Additionally, the

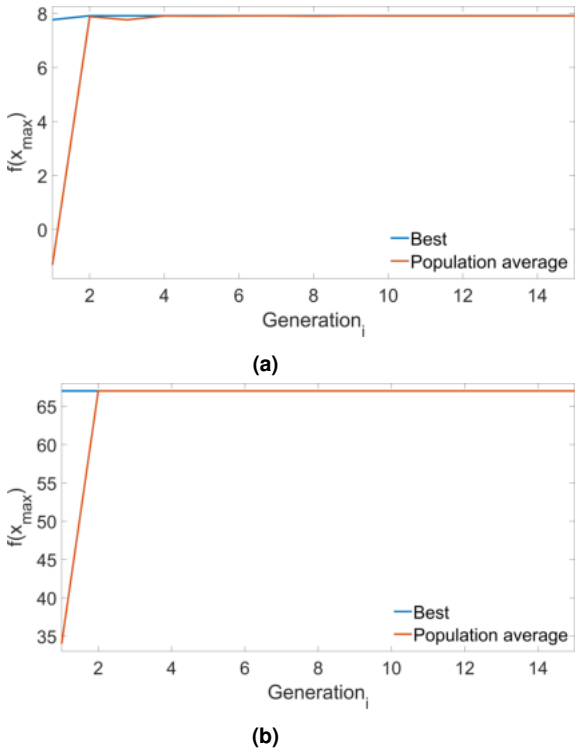


Fig. 2. Plot of the best and average values as a function of generation. The algorithm converged in the early generations. (a) Function f_1 and (b) Function f_2 with 15 generations on *Harmony*

generation counter is incremented at the end. The mutation is applied using the R_y gate, which consists of two parameters: the qubit of the $i-th$ individual to which a certain rotation will be applied, and the degrees of rotation, assigned according to Table 1.

In [22] explained that " Algorithm 2 provides the required rotation angle that will eventually modify the amplitude probability to change individual qubits of the quantum chromosomes.

The algorithm relies on Table 1 to select the appropriate angle for its use in Eq. 12; this updates the corresponding qubit in the quantum chromosome". The original strategy proposed in [10] to rotate the angle was modified as shown in Table 1:

$$\left[\alpha'_x \beta'_x\right] = \begin{bmatrix} \cos(\Delta\theta_i) & -\sin(\Delta\theta_i) \\ \sin(\Delta\theta_i) & \cos(\Delta\theta_i) \end{bmatrix} \begin{bmatrix} \alpha_x \\ \beta_x \end{bmatrix}. \quad (12)$$

For example, the way that we adapted the quantum inspiration was to take only the $i-jth$ values and compare them, assigning a rotation based on said comparison, trying to ensure convergence towards a better individual. Suppose you have the best individual $b_i = 10111110011$ and the new individual obtained is $x_i = 10110110110$ plus their respective decimal values are $\mathbf{X_i}$ and $\mathbf{B_i}$. If we will be taken $x_{ij} = 1$ and $b_{ij} = 1$, with $i = 1$ and $j = 1$. $f(\mathbf{X_i})$ and $f(\mathbf{B_i})$ are compared, using the Table 1, if $f(\mathbf{X_i}) \geq f(\mathbf{B_i})$, it will assign $\Delta\theta_i = 0.025\pi$ otherwise 0.005π.

3 Platform Amazon Web Service

AWS platform is designed to connect companies and customers who use and develop software for various purposes. One development within the AWS platform is Amazon Braket which was designed for research and applications in the area of quantum computing. D-Wave was the first device available in the launch of this platform on August 12, 2020. At present, it offers access to many quantum hardware development companies. The Amazon Braket name derives from the quantum mechanics term "bra-ket".

Currently, it provides access to quantum hardware developed by the companies listed in Table 2. By using this service, you can explore and design quantum algorithms such as Grover, Bernstein-Vazirani, and Deutsch-Jozsa, among others, execute them in different quantum simulators, run them on different quantum computers, and even demonstrate concepts about quantum computing.

Furthermore, it integrates Python notebook environments as well as MATLAB and PennyLane platforms. The way of working is in 3 stages: build, test, and execute. In addition, Amazon Braket enables the storage of all execution results and the running of algorithms without the requirement for individual configurations to specific device providers.

That is, it is enough to indicate the name of the quantum device that will be used. This makes it a convenient and easy-to-use platform for quantum computing. However, it is important to carefully review the available features of each

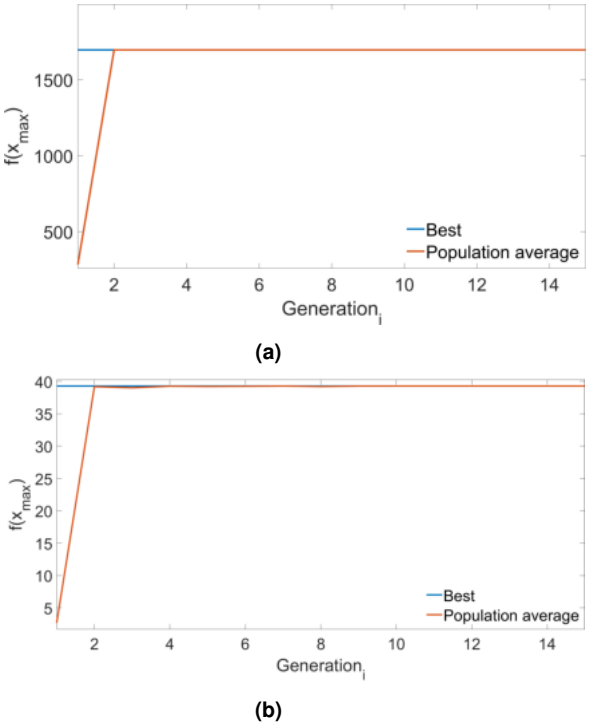


Fig. 3. Plot of the best and average values as a function of generation. The algorithm converged in the early generations. (a) Function f_3 on *Harmony* and (b) Function f_4 on *Simulator SV1* with 15 generations, respectively

company, since some devices do not support certain quantum operations.

This is because the supplier companies constantly update their quantum devices, and it takes time for this information to reach AWS. The circuit model is currently used as the foundation for representing and executing quantum operations by several well-established universal quantum computing platforms.

Quantum operations are depicted as quantum gates in this model, and computations are arranged as a series of these gates. Some quantum computing platforms that implement this programming paradigm include “IBM Quantum Experience” [26], “Google’s Sycamore” [1], and “Rigetti’s quantum computing platform” [12]. However, with AWS it is possible to have access to different kinds of hardware quantum. In [22] it is detailed in Table 2 form some characteristics of quantum devices available with this platform.

Furthermore, it also explains the offered access to three simulators, SV1, DM1, and TN1, which use AWS resources. SV1 is a tensor network simulator with 50 qubits, DM1 is a vector state simulator with 34 qubits, and TN1 uses a density matrix with 17 qubits.

It is pertinent to mention that quantum gates work in the same manner regardless of the programming language used, whether it is MATLAB or Python. Qubits are manipulated by quantum gates through the use of pulses that are sent to them.

As a result of these pulses, a wave is generated that is triggered by a signal connected to an input port on the hardware. Different parameters make up a signal such as amplitude, frequency, and duration.

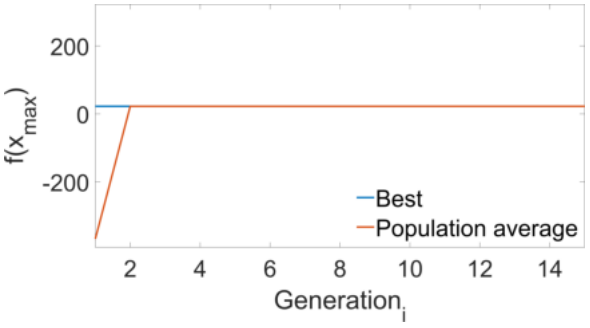
A quantum circuit is referred to as a quantum task. In this work, the quantum tasks represent the initial population operator, mutations, and crossovers. Each of these genetic operators is run for n generations (iterations) per execution to determine which result is most likely to be obtained.

Furthermore, for each quantum task, a certain number of measurements (shots) must be taken to mitigate error. The required number of shots will vary depending on the chosen quantum device. This refers to obtaining the measurement.

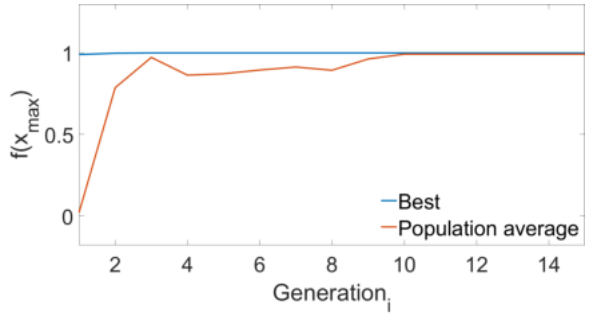
Although, the number of shots can be customized to suit your needs and error mitigation. The Amazon website recommends a minimum of 2,500 shots per quantum task for the *Aria-1* device. In particular, the executions in MATLAB had 100 shots, while the experiment with Python had 1,000.

This number of shots was used to stay within the project budget. AWS is a versatile platform that has grown and adapted to the needs of developers and emerging technologies. This company provides a variety of services, including the management of large databases, with high security until their processing. In addition to the accessibility of high-performance computing, such as machine learning and quantum computing. However, it is important to review the costs involved in each of these tools.

ISSN 2007-9737



(a)



(b)

Fig. 4. Plot of the best and average values as a function of generation. The algorithm converged in the early generations. (a) Function f_5 and (b) Function f_6 with 15 generations on simulator SV1

3.1 Implementation of QGA on Quantum Device in AWS

Referring to the design explained in Section 2.3, the resulting quantum circuits are depicted in Figure 1. For the first quantum population, we applied the *Hadamard* quantum gate described in Eq. 7. To achieve the quantum mutation and crossover processes, we used the $R_y(\theta)$ quantum gate described in Eq. 12.

The characteristics of QGA are taken from [22] where is used an array of 11 qubits to correspond to a single quantum chromosome. The population is constituted of 10 individuals.

A sequence of $m = 11$ qubits is defined as a quantum chromosome q_i, where $i \leq 10$. Therefore, we work with size chains of 11, each containing 10 elements. The precision was determined by the number of qubits available. In this case, 11 qubits were utilized. Among the devices listed in Table 2, only the *Harmony* from *IonQ* was accessible for MATLAB.

For Python, in addition to the *Harmony* device, *Aria-1* was also available. The domain was chosen for values of x between $[0.00, 127.00]$. To ensure consistency in both implementations, a quantization method was employed to obtain decimal values, and they were represented with 4 decimal places after the decimal point.

The implementation of MATLAB with a specific quantum device is facilitated by the development of a library called *'quantum'*, which was introduced in 2023. The connection procedure is initiated through the command exemplified in Code3.1. It is noteworthy that the establishment of this connection necessitates the creation of an Amazon Web Services account. This applies to both Python and MATLAB, although the process differs for each.

```
loadenv('awsAccount.env')
region = "us-east-1";
bucketStoragePath = "s3://amazon-braket-name
    Bucket/nameFolder/";
deviceARN="arn:aws:braket:us-east-1::device/
    qpu/ionq/Harmony";
device = quantum.backend.QuantumDeviceAWS
    (devARN, Region=region, S3Path=
    bucketStoragePath);
% Quantum circuit example
qcExample = quantumCircuit(HGate(1));
taskExample = run(qcExample, device);
wait(taskExample);
measure = fetchOutput(taskExample);
% The next line shows results in a table from
table(measure.Counts, measure.MeasuredStates,
    VariableNames=["counts","States"])
```

One of the differences between MATLAB and Python, as demonstrated in Code 3.1, is that Python requires only a single line of code to set the session values and select the quantum device after installing the Amazon Braket library. An example is shown in Code 3.1.

```
#Required packages
!pip install amazon-braket-sdk
!pip install boto3
!pip install --upgrade amazon-braket-sdk
!pip install --upgrade amazon-braket-schemas

from boto3 import Session
from braket.aws import AwsDevice, AwsSession
from braket.circuits import Circuit
from braket.simulator import BraketSimulator
```

```
#Simulator
from braket.devices import LocalSimulator

# Use the awsAccount name _and_ region
session=Session(
aws_access_key_id='ID_awsAccount',
aws_secret_access_key='access_key_awsAccount',
region_name='us-east-1')
# Establish a Braket session with Boto3
aws_sessionQPU = AwsSession(boto_session
=session)
# Any QPU device with the previously
# initiated AwsSession should be instantiated.
device_arn =
'arn:aws:braket:us-east-1::device/qpu/ionq/
Aria-1'
device = AwsDevice(device_arn,
aws_session=aws_sessionQPU)
```

On the other hand, with MATLAB, it is necessary to declare different variables for correct functionality, as shown in Code 3.1 where the `device` variable represents the chosen quantum device, while the `taskExample` variable refers to the quantum task design (quantum circuit). For better clarity on the connection in MATLAB, the following steps are provided below:

1. Commence by obtaining and installing the *'quantum'* library. It is important to note that this library is compatible with MATLAB versions commencing from 2023a.
2. To facilitate daily task execution, create an AWS account of the IAM type. This process entails:
   (a) Authorizing another user and setting up an "access key" from the Summary section.
   (b) Giving the authorization for using Amazon Braket services.
   (c) Handling permissions for the cloud-based storage solution (bucket S3).
3. Document the login credentials in a text file with the format "filename.env". With the next structure:
   (a) `AWS_ACCESS_KEY_ID= **ACCESS IDENTIFIER GENERATED**`
   (b) `AWS_SECRET_ACCESSS_KEY= **PROVIDED ACCESS KEY**`
   (c) `AWS_DEFAULT_REGION= **REGION OF THE QUANTUM DEVICE**`
4. Start with the `loadenv` command to load the contents specified by the "filename.env".
5. Generate an instance of "Bucket" labeled so that it commences with `amazon-braket-`
6. Inside this instance (bucket), create a repository to keep the outcomes of the computations.
7. Extract "Copy S3 URI" located in the repository generated in the bucket. This will be our "bucketStoragePath".
8. Ensure that the region configured for the bucket and the quantum hardware are identical.
9. Look into the available AWS quantum devices and simulators.
10. Choose one quantum device and obtain it "Device ARN". This value should be written in Code 3.1 to the variable "deviceARN".
11. Implement the Code 3.1 to initiate the connection to MATLAB.

Table 7. The best element obtained only in f_4 and f_6 with 3 runs of 10 generations in *Aria-1* the others functions are marked with '-' and 40 runs with 15 generations in the Local simulator on Python

Function f_n	AWS Device Python	Local Simulator Python
f_1	-	**7.9153**
f_2	-	**68.4705**
f_3	-	1,856.0263
f_4	**39.2008**	**39.2819**
f_5	-	**22.4994**
f_6	**0.9998**	**0.9901**

One essential component for storing the outcomes and requirements of our operations is the "Amazon S3 bucket" (Amazon Simple Storage Service). This service facilitates the storage of data as objects in a bucket, with a maximum storage capacity of 5 TB. It allows for the storage of various

Table 8. The mean, standard deviation, and median of the better elements obtained on 40 runs with 15 generations on the Local simulator on MATLAB and Python

f_n	Local Simulator MATLAB			Local Simulator Python		
	Mean	SD	Median	Mean	SD	Median
f_1	**7.9099**	0.0259	7.9167	**7.6636**	0.7396	7.8710
f_2	**68.4592**	0.1499	68.5000	66.9243	2.1947	67.6314
f_3	1867.5001	3.8520	1868.9434	1486.0648	277.0820	1537.3369
f_4	**39.1616**	0.4689	39.2826	38.3043	1.7572	39.1145
f_5	**22.4999**	0.0001	22.5	21.6097	4.9062	22.3917
f_6	**0.9998**	0.0002	0.9999	**0.5754**	0.3784	0.6355

file types, including videos, text files, and Braket task results.

In contrast, Python does not require manual configuration for generating necessary instances for storage; it automatically does so using the SageMaker tool. Therefore, the use of these tools should be periodically monitored.

The quantum task process is maintained in the same way in both languages because it is carried out internally by AWS. A task, which can be a quantum circuit or a quantum register, is defined and sent to the device for execution. In certain instances, the task is placed in a queue and held until the quantum device or simulator is prepared to receive it.

Following the submission of jobs to a quantum device, third-party companies with quantum computers process the tasks. Upon completion in MATLAB, the results are securely streamed to an "S3 bucket". Effective monitoring and management of all tasks can be performed on the "Quantum Tasks" page through the "Amazon Braket console".

3.2 Cost Analysis

Amazon Braket is currently an excellent option for diving into quantum computing, especially for countries with limited accessibility to different quantum hardware. However, it is important to consider the available budget to make the most of this tool. It is crucial to know the concept of a quantum algorithm for this reason.

In Section 2.2, it is explained that a quantum algorithm is a set of quantum gates that perform a specific task. However, we are in the NISQ era, where noise affects the consistency of results. This noise makes a single execution insufficient to guarantee accuracy due to the probabilistic nature of quantum outcomes.

Therefore, multiple measurements are required to statistically reduce the effect of variability caused by noise. In Amazon Braket, it is possible to configure the number of shots per quantum task according to specific needs, depending on the selected device. For example, with *Aria-1*, it recommends 2,500 shots.

In Table 3, we present the cost of each shot per quantum device and quantum tasks. It is crucial to emphasize that these devices are from multiple companies and utilize distinct technologies. Although our project did not focus on a specific technology, the type of technology used may be relevant for other types of problems. For instance, in our case, only the number of qubits available was relevant for a better representation of decimal numbers.

In our QGA process described in Subsection 2.3, we can observe two quantum tasks. The first task involves obtaining an initial quantum population using the Hadamard gate Eq. 7, while the second task involves executing mutation and crossover applied through by the quantum gate $R_y(\theta)$ Eq. 12, where θ is set using Table 1.

The final cost of each experiment is presented in Table 4. As detailed in the implementation section, each execution requires at least two other AWS tools, which incur additional costs for execution time, storage, and instance creation. Moreover, the cost varies based on the city chosen for tool execution.

Table 4 shows the costs derived from two experiments that consist of optimizing six different functions using MATLAB and Python. The executions in MATLAB had 100 shots, while the experiment with Python had 1,000. This number of shots was used to stay within the project budget. If the AWS recommendation of using 2,500 shots for more reliable measurements had been followed, the total cost for both experiments would have been USD$84,110.1, in contrast to the USD $5,014 spent. The costs associated with AWS encourage the search for different options to work

with quantum devices. One possible proposal is the purchase of a quantum device.

For example, purchasing a device "Gemini Mini"[3] with a price of around $5,000, which is equipped with 2 qubits and based on the theory of nuclear magnetic resonance (NMR) [24]. However, this option is limited compared to Amazon Braket due to the restricted number of available qubits and the reliance on a single technology.

Therefore, the choice of quantum hardware will depend on the research objectives and the available budget. In this sense, Amazon Braket offers a wider range of quantum technology options.

4 Experiments and Results

This section presents the experiments and results implementing the QGA on a single-variable optimization problem using two platforms available to connect with quantum devices on AWS. The first platform utilizes the *'quantum'* library for MATLAB, and the second uses the Python programming language through "Amazon Braket". Finally, results from local simulators enabled on both platforms are compared. The QGA is assessed on different test functions with different levels of complexity to evaluate the strengths and weaknesses of the QGA. Table 5 presents the selected test functions with their respective global maximum to be found.

The goal is to determine if there is a difference in performance, specifically regarding the time required, and to ascertain if there is a variance in the results obtained from the function optimization problem. The assumption is that since we employ the same algorithm for both platforms, any deficiency should be evident in both sets of results.

The objective of QGA is to determine the value of x that maximizes the functions shown in Table 5. Individuals were initially encoded in binary format and then converted to decimal values, as described in subsection 3.1. Two experiments were conducted. The first experiment aimed to define and observe the requirements and behavior of the connection with the MATLAB platform, and the second with Python. In both experiments, the precision of the QGA was examined.

[3] spinquanta.com/products-solutions/gemini

4.1 Experiment 1: MATLAB Implementation

The experiment for the MATLAB implementation involved running fifteen generations per test, with 100 shots for each quantum circuit. The *Harmony* device with 11 qubits was used for the first three test functions, and the *Amazon SV1* simulator was used for the last three test functions.

These adjustments were made to stay within the project budget, and due to the variability of costs in available quantum devices shown in Table 3, the number of runs in this test was 10. For comparison with the local simulator, 40 executions with 15 generations were carried out.

From this experiment, an average of all the best results obtained was calculated, as well as their standard deviation, as shown in Table 8 in comparison subsection 4.3 between local simulators. In the case of the MATLAB experiment with a quantum device, only the final solution of the 10 runs was obtained. Table 6 displays the final results.

The first column presents the best tuple $(x, f(x))$ obtained using the *Harmony* quantum device, and the second column shows the best result obtained with the quantum simulator. The value close to the maximum is in bold. It is important to note that both used the 'quantum' package.

In table 6, a relevant feature is presented regarding the approximation of the value of x. Quantization, which depends on the number of available qubits, is used to enhance the approximation when converting bits to decimal numbers. With 11 qubits, it is evident that the decimal values of x differ from the value indicated in Table 5. Nevertheless, in this experiment, this variance did not have a significant impact on the QGA objective.

Table 6 shows the MATLAB approximations performed on the *Harmony* device and Local Simulator. The executions on the Local Simulator were on an Intel Core i7-9750 Processor (2.6 GHz) with 16 GB of RAM running Windows 11 Home with MATLAB 2023.

ISSN 2007-9737

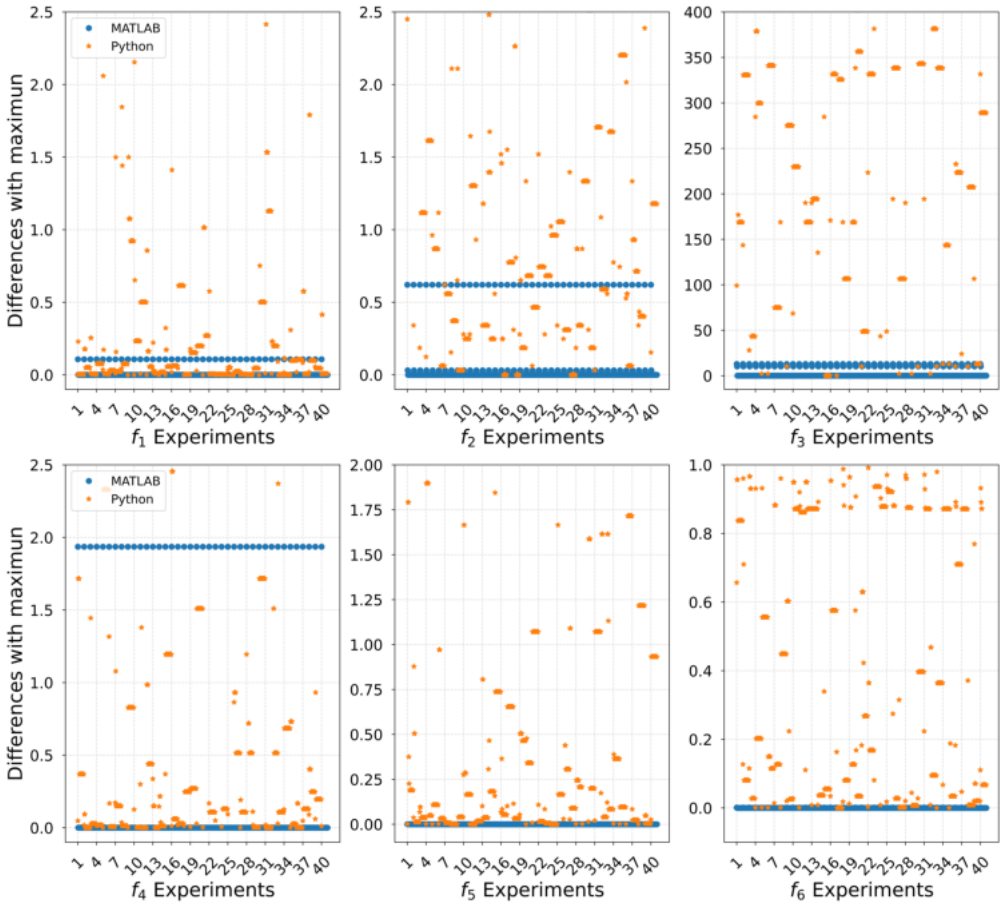


Fig. 5. Differences between the best elements found and the maximum of all functions

Regarding the behavior of the Quantum Genetic Algorithm (QGA) on MATLAB, the results depicted in Figures 2, 3, and 4 illustrate that the QGA requires a few generations to reach optimal or near-optimal values.

They also have an average that describes how the function values near the optimum or even the optimum value have been found since early generations. This is a promising outcome, especially compared to simulation results from previous studies such as [2], [13] and [17] which predict favorable results.

4.2 Experiment 2: Python Implementation

In the second experiment, we ran ten generations per test for the Python implementation of the QGA, using 1,000 shots on f_4 and f_6 with the *Aria-1* device, which contains 25 qubits. These limitations were set to stay within the budget. The number of executions was limited to three.

For the local simulator with the same characteristics mentioned in the previous experiment, we had 40 algorithm executions (runs). A comparison of the simulator run and the data obtained from the *Aria-1* quantum device run is shown in Table 7.

In Table 7, we can see that the results on the quantum device are near the maximum, while the local simulator also shows greater accuracy in the other functions. The closest values are highlighted in bold.

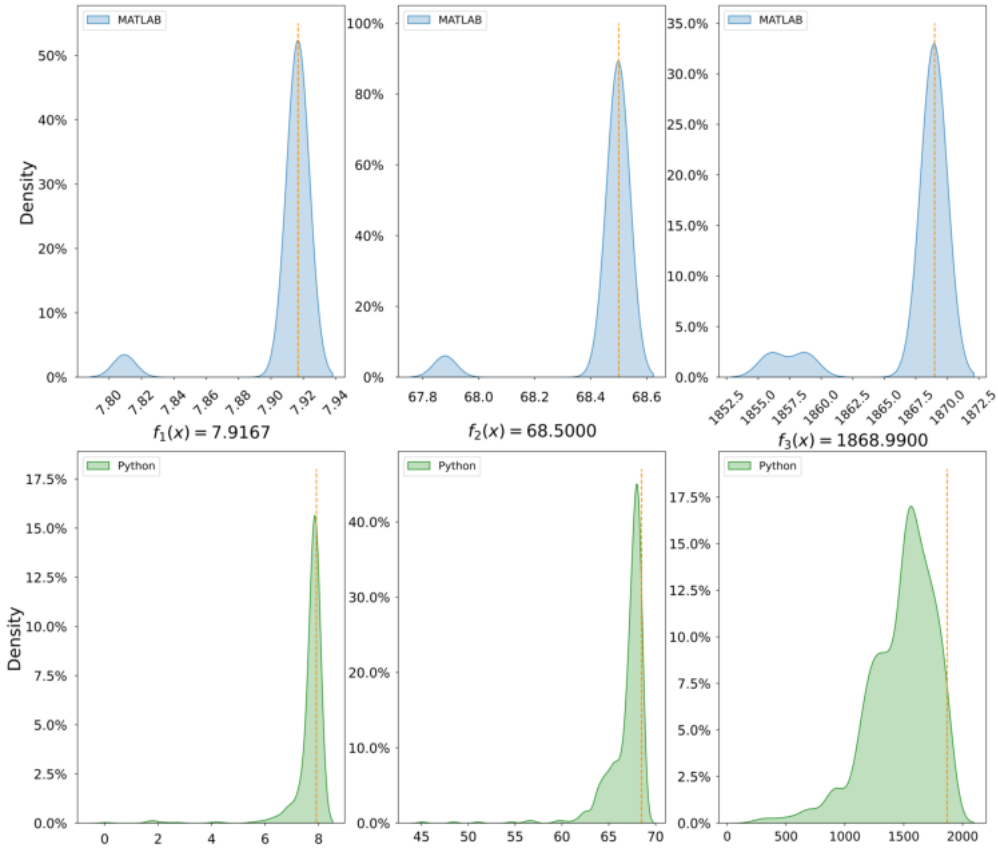


Fig. 6. Distribution of maximum findings for functions f_1 - f_3. Figures top MATLAB, bottom Python

4.3 Comparison of Local Simulator

Finally, we compare the two simulations made with the QGA versions. One simulation used the MATLAB *'quantum'* package and the other used *Amazon Braket* in Python with our computing resources (Intel Core i7-9750 Processor (2.6 GHz) with 16 GB of RAM running Windows 11 Home with MATLAB 2023 and Python 3.11) to simulate quantum processing. Table 8 shows the mean, standard deviation, and median.

The MATLAB data shows a lower standard deviation compared to Python, as well as a good mean approximation toward the target. In Table 8, it is possible to observe that the median results closest to the maximum for Python are with f_1 and f_6, which also have a lower deviation. In the case of MATLAB, f_1, f_2, f_4, f_5, and f_6 test functions show good performance, data that are shown in bold.

In Figure 5, the difference between the maximum value to be obtained, and the values generated by QGA on both platforms is shown where each function exhibits different behaviors. Additionally, the behavior with both libraries shows significant differences, which are consistent with Table 8. A high dispersion is observed for the case of Python (marked with a star).

The observed wiggly points are due to the random nature of the algorithm in every run, where each execution begins with a new quantum population that evolves after each generation (15). With MATLAB, the jumps are more noticeable as it approaches the last generation of each run, containing better elements and resulting in almost zero difference.

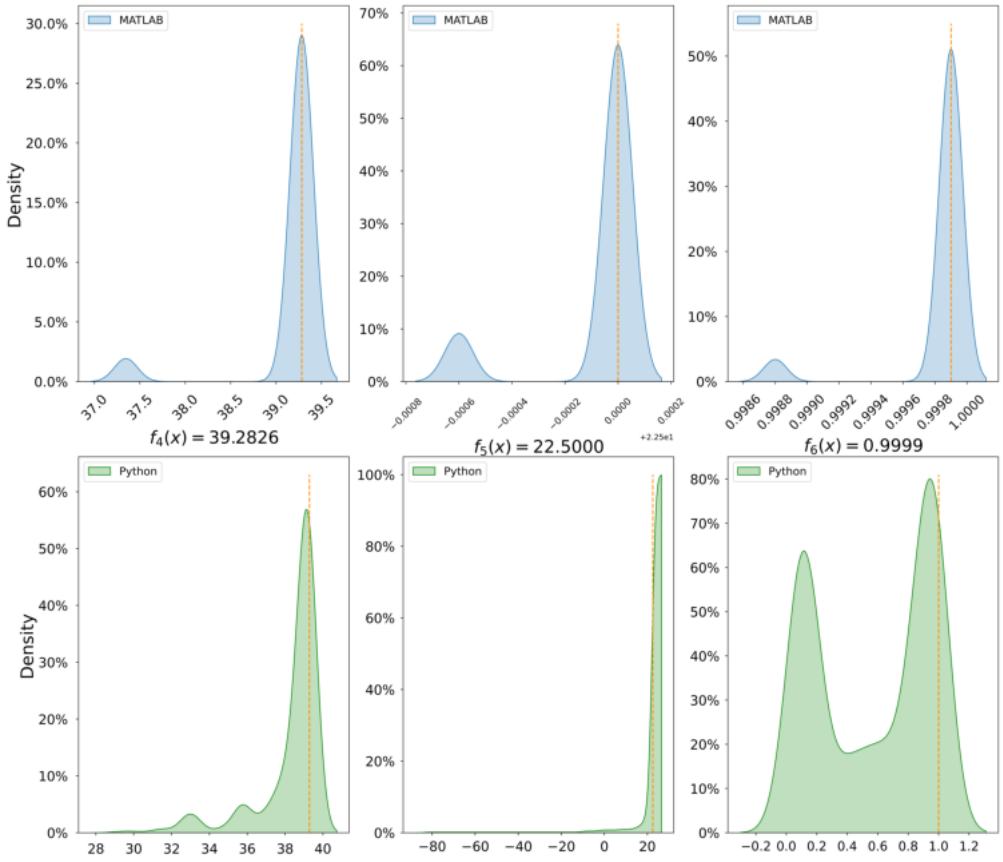


Fig. 7. Distribution of maximum findings for functions f_4 - f_6. Figures top MATLAB, bottom Python

In contrast, with Python, only some runs show improvement after each generation. For example, for test functions f_1, f_4, and f_5, a behavior similar to that of MATLAB can be seen, but not for the other functions.

The distribution of the total of the best individuals obtained from the local simulator executions is shown in Figure 6 and Figure 7. Results obtained with Python are represented in green, while those obtained with MATLAB are in blue, with a dotted line indicating the maximum required value.

Both Figures (6 and 7) demonstrate the precision of MATLAB. Although Python and MATLAB both show that the highest percentage of the values obtained are close to the maximum required, it is evident that Python produces results that are further away from the maximum required value compared to MATLAB.

On the other hand, Table 9 shows the execution times with MATLAB using *Harmony* and with Python using *Aria-1*, as well as the times with the local simulator, available on both platforms. Additionally, Table 9 shows the particular time to generate the quantum population and the quantum mutation (quantum tasks), data obtained from the record that AWS makes on the Amazon Braket console.

Additionally, it is possible to observe a significant difference in the time described in Table 9. The average time results shown in Table 9 were obtained from approximately 583 quantum tasks with 100 shots in MATLAB. For Python results, approximately 534 tasks with 1,000 shots. It is imperative to provide additional information

Table 9. Mean, standard deviation, and total time of the QGA execution with MATLAB and Python on a quantum device (3 and 10 runs with 15 and 10 generations respectively), and on a local simulator (40 runs with 15 generations each)

	MATLAB - *Harmony*			**MATLAB - *Local Simulator***		
	Mean (s)	SD	**Total Time(s)**	Mean (s)	SD	**Total Time(s)**
Quantum population	1,320	0.0206	-	-	-	-
Quantum mutation	2,100	0.0563	-	-	-	-
Total mean	2,040	0.0545	-	18.4259	9.5568	-
-	-	-	1,354.8	-	-	110.5556
	Python - *Aria-1*			**Python - *Local Simulator***		
Quantum population	36.5996	23.5761	-	-	-	-
Quantum mutation	36.5996	23.5761	-	-	-	-
Total mean	73.1993	47.1523	-	37.5076	0.0079	-
-	-	-	512.3955	-	-	225.0478

on operating hours to access specified quantum devices, such as *Harmony* and *Aria-1*.

These devices are available Monday through Friday from 12:00:00 to 03:00:00 UTC (Coordinated Universal Time). The above detail is significant since it introduces an additional waiting time to the total duration of the execution, a factor that is not considered in the current analysis because this time is independent of the chosen platform/programming language.

Likewise, AWS offers accessibility mechanisms such as *Braket Direct*, which facilitates time-bound requests for the use of specific quantum devices, and *Hybrid Jobs*, which encompasses hybrid quantum tasks.

The results shown in Figures 5, 6, and 7 depict the precision of the QGA in each test function. In that sense, statistics on the differences from the established maximum were obtained. A noticeable difference can be observed in test function f_3 due to its multimodal behavior.

However, in general, we can see that for the rest of the test functions, the range of differences is much closer to the global maximum. The behavior of the QGA for the optimization of single-variable multimodal functions shows results consistent with previous studies, such as those by [13, 17, 18].

These studies conducted on simulators predict a good performance of the QGA. One of the significant advantages is the reduced number of generations or individuals required to find the maximum, resulting in substantial time and computational resource savings.

This performance is observed even in devices that, to date, contain a limited number of qubits. Two notable aspects of this work are time and cost, as access to a quantum device depends on the stable connection with AWS and the connection between AWS and the company owning the quantum computer.

Additionally, the queue of jobs waiting for each device and their established response times are crucial factors. These factors are important to consider when working with AWS, as the availability of various devices may be affected.

For example, when attempting to execute a task on certain devices, such as *Aquila*, the device may be suspended by the company, resulting in execution errors. Additionally, internet availability is another crucial factor to consider.

5 Conclusion and Future Work

This study demonstrated the feasibility of implementing Quantum Genetic Algorithms (QGA) implemented via Amazon Braket on MATLAB and Python platforms. Moreover, it highlited relevant details, such as the dependence on the stability of the connection with AWS and the response times of the quantum computers' owner companies for accessing quantum devices.

The queue of jobs and internet availability also play crucial roles. These factors must be considered when working with AWS, as device availability may vary. For instance, certain devices might be suspended by their companies, leading to execution errors.

On the other hand, the QGA performance in optimizing single-variable multimodal functions is consistent with previous studies conducted on simulators. The QGA requires fewer generations or individuals to find the maximum, thus saving time and computational resources. This efficiency is maintained even on devices with a limited number of qubits. Quantum metaheuristics represent a promising advancement over classical metaheuristics, particularly for tackling complex high-dimensional problems.

Classical metaheuristics have demonstrated remarkable results, and the significant resources required emphasize the necessity of further exploration into quantum approaches. These approaches leverage fundamental quantum physics features such as superposition and entanglement, potentially leading to exponential improvements in execution time and resource efficiency.

Despite the challenges posed by the NISQ era, including quantum errors, noise, limited qubit availability, and hardware variability, this study provides a practical initial application of quantum devices, yielding the expected results.

Furthermore, the use of simulations accentuates the utility of widely adopted programming languages, such as Python and MATLAB, making quantum computing more accessible to the scientific community. This accessibility is crucial for advancing research and development in this rapidly evolving field.

Future work should explore multimodal optimization problems using different technologies to determine if certain technologies are more suitable for specific problems. This approach will help assess the comparative performance of various technologies.

For initial forays into quantum computing or when analyzing various quantum algorithms without a budget or clear time frame, cloud-based tools such as Qiskit and Pennylane are recommended.

Errors on the AWS platform can be costly in terms of money and time. AWS is best suited for developing well-established problems where exact execution costs can be calculated or when a large budget is available.

In conclusion, while the current state of quantum hardware presents several challenges, the continued development and expansion of quantum devices on platforms like Amazon Braket promise to enhance quantum computing capabilities.

Future research should focus on improving QGAs and exploring their applications across different technologies to address complex optimization problems more effectively.

Acknowledgments

We thank the National Council of Humanities, Sciences, and Technologies (CONAHCYT) and Instituto Politécnico Nacional for supporting our research activities through project numbers CF-2023-I-108 and SIP20240164, respectively. This work was funded by the Red-CI Baja-AWS project of the Government of the State of Baja California.

References

1. **AbuGhanem, M., Eleuch, H. (2023).** Experimental characterization of Google's sycamore quantum AI on an IBM's quantum computer. DOI: 10.2139/ssrn.4299338.

2. **Acampora, G., Chiatto, A., Vitiello, A. (2023).** Genetic algorithms as classical optimizer for the quantum approximate optimization algorithm. Applied Soft Computing, Vol. 142, pp. 110296. DOI: 10.1016/j.asoc.2023.110296.
3. **Ballinas, E., Montiel, O. (2023).** Hybrid quantum genetic algorithm with adaptive rotation angle for the 0-1 knapsack problem in the IBM Qiskit simulator. Soft Computing, Vol. 27, No. 18, pp. 13321–13346. DOI: 10.1007/s00500-022-07460-7.
4. **Blum, C., Roli, A. (2003).** Metaheuristics in combinatorial optimization: Overview and conceptual comparison. ACM Computing Surveys, Vol. 35, No. 3, pp. 268–308. DOI: 10.1145/937503.937505.
5. **Butenko, S., Pardalos, P. M. (2014).** Numerical methods and optimization: An introduction. CRC Press.
6. **De-los-Cobos, S. G., Goddard-Close, J., Gutiérrez-Andrade, M. A., Martínez-Licona, A. E. (2010).** Búsqueda y exploración estocástica. Universidad Autónoma Metropolitana.
7. **Dey, S., Bhattacharyya, S., Maulik, U. (2019).** Quantum inspired metaheuristics for image analysis.
8. **Gill, S. S., Kumar, A., Singh, H., Singh, M., Kaur, K., Usman, M., Buyya, R. (2022).** Quantum computing: A taxonomy, systematic review and future directions. Software: Practice and Experience, Vol. 52, No. 1, pp. 66–114. DOI: 10.1002/spe.3039.
9. **Han, K. H., Kim, J. H. (2000).** Genetic quantum algorithm and its application to combinatorial optimization problem. Proceedings of the 2000 Congress on Evolutionary Computation, Vol. 2, pp. 1354–1360. DOI: 10.1109/CEC.2000.870809.
10. **Han, K. H., Kim, J. H. (2002).** Quantum-inspired evolutionary algorithm for a class of combinatorial optimization. IEEE Transactions on Evolutionary Computation, Vol. 6, No. 6, pp. 580–593. DOI: 10.1109/TEVC.2002.804320.
11. **Haupt, R. L., Haupt, S. E. (2004).** Practical genetic algorithms. Wiley, Hoboken, New Jersey, USA.
12. **Karalekas, P. J., Tezak, N. A., Peterson, E. C., Ryan, C. A., da-Silva, M. P., Smith, R. S. (2020).** A quantum-classical cloud platform optimized for variational hybrid algorithms. Quantum Science and Technology, Vol. 5, No. 2, pp. 024003. DOI: 10.1088/2058-9565/ab7559.
13. **Lahoz-Beltra, R. (2016).** Quantum genetic algorithms for computer scientists. Computers, Vol. 5, No. 4, pp. 24. DOI: 10.3390/computers5040024.
14. **Malossini, A., Blanzieri, E., Calarco, T. (2008).** Quantum genetic optimization. IEEE transactions on evolutionary computation, Vol. 12, No. 2, pp. 231–241. DOI: 10.1109/TEVC.2007.905006.
15. **Marler, R. T., Arora, J. S. (2004).** Survey of multi-objective optimization methods for engineering. Structural and multidisciplinary optimization, Vol. 26, pp. 369–395. DOI: 10.1007/s00158-003-0368-6.
16. **Mitchell, M. (2001).** An introduction to genetic algorithms. Bradford, Cambridge, Massachusetts, USA.
17. **Montiel, O., Rubio, Y., Olvera, C., Rivera, A. (2019).** Quantum-inspired acromyrmex evolutionary algorithm. Scientific Reports, Vol. 9, No. 1. DOI: 10.1038/s41598-019-48409-5.
18. **Montiel-Ross, O. H. (2020).** A review of quantum-inspired metaheuristics: Going from classical computers to real quantum computers. IEEE Access, Vol. 8, pp. 814–838. DOI: 10.1109/ACCESS.2019.2962155.
19. **Narayanan, A., Moore, M. (1996).** Quantum-inspired genetic algorithms. pp. 61–66. DOI: 10.1109/ICEC.1996.542334.

20. **Quiroz-Castellanos, M., de-la-Fraga, L. G., Lara, A., Trujillo, L., Schütze, O. (2023).** Numerical and evolutionary optimization 2021. Mathematical and Computational Applications, Vol. 28, No. 3, pp. 71. DOI: 10.3390/mca28030071.
21. **Rieffel, E. G., Polak, W. H. (2011).** Quantum computing: A gentle introduction.
22. **Rosales-Alvarado, S. S., Montiel, O., Orozco-Rosas, U., Tapia, J. J. (2024).** Developing a quantum genetic algorithm in MATLAB using a quantum device on AWS. Springer Nature Switzerland, Cham, pp. 111–127. DOI: 10.1007/978-3-031-53713-4_10.
23. **Rubio, Y., Olvera, C., Montiel, O. (2021).** Quantum-inspired evolutionary algorithms on IBM quantum experience. Engineering Letters, Vol. 29, No. 4, pp. 1573–1584.
24. **Sels, D., Dashti, H., Mora, S., Demler, O., Demler, E. (2020).** Quantum approximate Bayesian computation for NMR model inference. Nature machine intelligence, Vol. 2, No. 7, pp. 396–402.
25. **Sun, Y., Xiong, H. (2014).** Function optimization based on quantum genetic algorithm. Research Journal of Applied Sciences, Engineering & Technology, Vol. 7, No. 1, pp. 144–149.
26. **Wootton, J. R., Harkins, F., Bronn, N. T., Vazquez, A. C., Phan, A., Asfaw, A. T. (2021).** Teaching quantum computing with an interactive textbook. 2021 IEEE International Conference on Quantum Computing and Engineering (QCE), pp. 385–391. DOI: 10.1109/QCE52317.2021.00058.
27. **Yanofsky, N. S., Mannucci, M. A. (2009).** Quantum computing for computer scientists.

Article received on 23/05/2024; accepted on 18/07/2024.
**Corresponding author is Oscar Castillo.*