ISSN 2007-9737

Hybrid Quantum Genetic Algorithm for the 0-1 Knapsack Problem in the IBM Qiskit Simulator

Enrique Ballinas, Oscar Montiel

Instituto Politécnico Nacional,
Centro de Investigación y Desarrollo en Tecnología Digital,
Mexico

lballinas@citedi.mx, oross@ipn.mx

Abstract. In this work, a novel Hybrid Quantum Genetic Algorithm (HQGA) for the 0-1 Knapsack Problem (KP) is presented. It is based on quantum computing principles, such as qubits, superposition, and entanglement of states. The HQGA was simulated in the Qiskit simulator. Qiskit simulator is a platform developed by IBM that allows working with quantum computers at the level of circuits, pulses, and algorithms. The performance of HQGA is evaluated in three strongly correlated KP data sets, and computational results are compared with a Quantum-Inspired Evolutionary Algorithm (QIEA), a modified version of a QIEA (QIEA-Q), and a modified version of the HQGA (HQGA-Q). Experimental results demonstrate that the proposed HQGA can obtain the best solutions in all the KP data sets, and performs well on robustness.

Keywords. Quantum computing, quantum genetic algorithm, knapsack problem.

1 Introduction

Solving combinatorial optimization problems using classical exact algorithms becomes infeasible when the number of instances reaches a tractable limit for classical computation since search space for candidates' solutions tends to grow exponentially [48]. The complication surge when it is inevitable to handle larger values of instances that exceed this limit, which is very common in real-life applications. For example, in energy systems optimization, where there are several challenges for classical computation that can be tackle with quantum computing, such as the problem of Heat exchanger network synthesis (HENS) where a simple HENS subproblem face the programmer with an NP-hard problem [2]; or in the financial market, the transaction settlement problem is also a good example [8]. At present, it is common to handle these problems using approximation algorithms; they have the characteristics of given an approximate solution to a particular kind of problem.

They search for a solution very close to the optimal one in polynomial time according to the sizes of their inputs instead of looking for the optimal solution, which can cause the search time grows exponentially [31]. They are divided into three types, heuristic, meta-heuristic, and hyper-heuristic. Heuristic methods are based on experience that allows finding a satisfactory solution in a reasonable time. Meta-heuristics are those algorithms that are one level higher than heuristics; that is, they are procedures that seek to find a solution to a problem using the least amount of computational resources than heuristic algorithms [11].

Hyper-heuristics can be viewed as search algorithms that explore the space of problem solvers. A hyper-heuristic is a heuristic search method that seeks to automate the process of selecting, combining, generating, or adapting several simple heuristics in order to solve a problem efficiently [11]. The importance of meta-heuristic algorithms was fully understood when the NP-completeness theory was established in 1971 [31]. This theory determined that many of the known problems in state of the art are intractable, which means

that it is not possible to find an optimal solution in a polynomial-time [10]. As a consequence, approximation algorithms were the best option to solve these kinds of problems. The approximation algorithms have experienced significant growth in the last years due to the development of areas such as data mining, bioinformatics, deep learning, and others; this caused a significant number of optimization problems to be born.

Quantum computing offers an exponential speedup for solving some problems that can take advantage of quantum phenomena in their algorithmic formulation. This feature is desirable for solving problems where classical and approximation algorithms cannot offer practical or viable solutions to complex problems that have grown very fast and become unsolvable.

The contribution of this paper is the design of a Hybrid Quantum Genetic Algorithm (HQGA) and its implementation in a quantum simulator (Qiskit IBM simulator). So far, there is no work in state-of-the-art where a Quantum Genetic Algorithm in the IBM Qiskit simulator can solve the Knapsack Problem (KP). There is only one work [29] where Adiabatic Quantum computation was implemented in the Qiskit simulator to solve the Binary Knapsack Problem (it is explained in section 2). Also, a quantum circuit was designed which allows to generate the initial population of qubits, with this, the diversity of the algorithm is expanded since when using qubits, the possible configurations increase exponentially just by adding a qubit to the quantum population. For example, with a population of 10 qubits we would have $2^{10} = 1024$ possible configurations, with 11 qubits $2^{11} = 2048$ possible configurations we would have.

A modified version of the HQGA (HQGA-Q), a QIEA, and a modified version of the QIEA (QIEA-Q) was also designed. HQGA-Q and QIEA-Q were designed to solve an impediment of the Qiskit simulator; in section 4 it will be explained. The HQGA, based on the statistical results of hundreds of tests performed, has proven to outperform the aforementioned quantum evolutionary algorithms. The main advantage of HQGA compared to state-of-the-art algorithms is the implementation of a quantum circuit in a quantum genetic algorithm, which takes advantage of one of the main characteristics of quantum computing, quantum parallelism, with this we can represent n possible configurations with just a quantum register of n qubits.

The organization of this paper is as Follows. Section 1 presents an introduction to approximation algorithms and the importance of meta-heuristics algorithms in solving combinatorial optimization problems. Section 2 presents the main works that use quantum meta-heuristics in solving the knapsack problem. Section 3 describes the main concepts involved in the development of this work. Section 4 shows the experiments carried out and their corresponding results in solving the 0-1 knapsack problem. The conclusions and future work are presented in section 5.

2 Related Work

The knapsack problem (KP) is a widely studied combinatorial optimization problem with NP-hard computational complexity [22, 14, 52, 26]. In the literature, there are several works that have solved the 0-1 Knapsack problem using different methods, such as, Binary Cuckoo Search Algorithm (CSA) [6], Firefly Algorithm (FA) [7], Ant Colony Algorithm (ACO) [42], Comparative study between Tabu Search Algorithm (TS), Sparse Search (SS), and Local Search [41], Comparative study between Simulated annealing algorithm (SA), Iterative local search, Genetic algorithm (GA), and Particle swarm optimization algorithm (PSO) [1], among others.

The methods mentioned above are evolutionary and non-evolutionary algorithms that have been widely used for solving combinatorial optimization problems, showing satisfactory results using conventional computers. However, it is possible to improve these results using quantum algorithms.

Quantum computing is a computational model that uses certain quantum mechanics concepts, such as superposition, entanglement, and qubits. When it is combined with genetic algorithms, quantum genetic algorithms are created [49, 15].

Quantum genetic algorithms have demonstrated the potential to tackle NP-hard problems, even showing better results than classical algorithms.

For example, in [17], a series of experiments using a Parallel Quantum-Inspired Genetic Algorithm, Quantum-Inspired Genetic Algorithm, and a Classical Genetic Algorithm are presented. The results demonstrate the Parallel Quantum-Inspired Genetic Algorithm's superiority over the other two solving the 0-1 Knapsack Problem. The use of parallel computing helps to improve the exploitation and exploration capabilities.

In [46], a Higher-order Quantum Genetic Algorithm to solve the 0-1 Knapsack Problem with 200, 500, and 1000 objects is proposed. The algorithm uses high-order quantum registers, which consists in dividing the register into sub-registers. This reduced the algorithm's execution time compared to the traditional Quantum Genetic Algorithm (QGA) and maintained the same precision, even increasing it in some registers sizes.

In [45], a Quantum Genetic Algorithm (QGA) to solve the Knapsack Problem is implemented. QGA uses an adaptive quantum gate to find the rotation angle's correct value, reducing the qubits states' amplitudes with respect to the previous results. The experimental results showed that the use of the adaptive quantum gate generates a fast local convergence by solving the Knapsack Problem with different quantities of objects.

Another type of meta-heuristic that has proven to be efficient in solving the NP-hard problems, especially the Knapsack Problem, is the Quantum Evolutionary Algorithms. For example, in [43] a Self-organizing Quantum Evolutionary Algorithm for Multi-objective optimization (MSQEA) that solves the Multi-objective Knapsack Problem is designed. The experimental results showed that the MSQEA could obtain solutions very close to the Pareto optimal front in a short time, in addition to generating larger sets of non-dominating points.

In [51] an Improved Quantum Evolutionary Algorithm (IQEA) is proposed; this algorithm's main feature is using the rotating quantum gate that generates a faster convergence and a better global search for solutions. The algorithm was compared against an Evolutionary Quantum Algorithm (QEA), demonstrating its superiority in efficiency and quality when solving the Knapsack Problem.

In [27], a Quantum Evolutionary Algorithm to solve the Quadratic Knapsack Problem (QKP) is designed. The qubits are initialized according to the density values (this value is obtained by dividing the sum of the gains of all objects and the object's weight) and are expressed in angles; this proposal was called Angle-expressed Quantum Evolutionary Algorithm (AQEA). AQEA was compared with a Classic Genetic Algorithm (CGA) and a Quantum Evolutionary Algorithm (QEA), demonstrating its effectiveness and better convergence when solving QKP with 100, 200, and 300 objects.

In [20], a Quantum-Inspired Evolutionary Algorithm to solve the Multidimensional Knapsack Problem is designed. Experiments were carried out with different repair functions: Simple, Random, and Sorted. The results showed that the Sorted repair function has a faster convergence time; however, having to sort the objects beforehand would be a great effort if the data set is huge.

One of the important characteristics of quantum computing is the versatility of the algorithm to be able to adapt to different meta-heuristics and search algorithms. For instance, in [24], a Diversification-based Quantum Particle Swarm Optimization Algorithm (DQPSO) to solve the Multidimensional Knapsack Problem is presented. DQPSO is based on the Quantum Particle Swarm Optimization Algorithm (QPSO) and a population-based diversification criteron, which generates better diversity than QPSO. The experiments were carried out using 30 instances showing the efficiency of DQPSO.

In [25], a Quantum Particle Swarm Optimization Algorithm with a preserving strategy (Diversity-preserving QPSO) to solve the Multidimensional Knapsack Problem is presented. The Diversity-preserving QPSO has a diversity strategy to update and maintain a good diversity in the population, and a method called Variable Neighborhood Descending (VND) which improves the search process to find the optimal solution. The results demonstrated the efficiency of the Diversity-preserving QPSO compared to the state-of-the-art algorithms.

In [34], an algorithm called Binary Quantum Inspired Gravitational Search Algorithm (BQIGSA) is presented; it combines the properties of the Gravitational Search Algorithm (GSA) and quantum algorithms; some experiments were carried out with the Max-one, Royal-road functions, and Knapsack Problem. The results were compared with the Binary Gravitational Search Algorithm (BGSA), Conventional Genetic Algorithm (CGA), Binary Particle Swarm Algorithm (BPSO), a modified version of the BPSO, a new version of the Binary Differential Evolution Algorithm, Quantum-Inspired Particle Swarm Algorithm, and three Quantum-Inspired Evolutionary Algorithms. For the problems mentioned above, the BQIGSA algorithm presented the best results.

In [18], A Binary Multi-scale Quantum Harmonic Oscillator Algorithm (BMQHOA) to solve the Knapsack Problem is designed. BMQHOA is inspired by the probabilistic interpretation of the wave function and using properties such as quantum tunnel effect avoids local optimum. Several experiments were carried out with the following algorithms: Binary Bat Algorithm (BBA), Binary Dragonfly Algorithm (BDA), Binary Particle Swarm Algorithm (BPSO) and Binary Particle Swarm with Gravitational Search Algorithm (BPSOGSA). The results showed the superiority of BMQHOA in precision, convergence, and stability compared to the algorithms mentioned above.

In [13], a Quantum-inspired Wolf pack Algorithm to solve the 0-1 Knapsack Problem is presented. The algorithm is based on the behavior of the wolf pack when hunting; this approach uses quantum gates to update the position of the solutions. Experiments were carried out with 100, 250, 500, and 1000 dimensions. Compared with other algorithms in the literature, the results showed the proposed algorithm's effectiveness, especially for large cases.

In [12], a Quantum Annealing Algorithm (QA) that uses parallel computing properties to solve the Multidimensional Knapsack Problem was introduced. QA is a technique derived from Simulated Annealing Algorithm (SA). The experiments with 500 objects showed that QA outperforms its non-parallel version.

We end this section with a work where Adiabatic Quantum Computing (AQC) was used to solve the Binary Knapsack Problem using the libraries of IBM Qiskit simulator [29]. AQC is considered a particular class of Quantum Annealing (QA), which uses quantum mechanics properties to solve optimization problems without restrictions. The results obtained show the quantum algorithm's effectiveness and a slight superiority compared to its classical counterpart.

3 Theoretical Framework

3.1 The Knapsack Problem

One of the most studied combinatorial optimization problems is the knapsack problem, which is computationally challenging because it is NP-hard. [14, 35, 22]. The problem consists in given a set of objects, each with a weight w_i and value p_i, to determine which objects should be part of a collection with the condition that the weight WX is less than or equal to a certain limit, and the total value PX is what largest possible [52].

The mathematical representation of the model is presented below:

$$\begin{aligned} &\text{maximize } f(x_1, x_2, \ldots, x_n) = PX = \sum_{i=1}^{n} p_i x_i, \\ &\text{subject to } WX = \sum_{i=1}^{n} w_i x_i \leq V, \end{aligned} \tag{1}$$

$x_j \in \{0, 1\}$, $j = 1, 2, \ldots, n$. Here $P = (p_1, p_2, \ldots, p_n)$, $W = (w_1, w_2, \ldots, w_n)$ represent the vector of values and the vector of weights of all objects respectively. V is the maximum capacity of the knapsack, $x_i = 1$ indicates that the object i is inside of the knapsack and $x_i = 0$ that it is not.

3.2 Quantum Computing

A classical computer uses bits to store information, whereas a quantum computer uses quantum bits or qubits. Qubits are a unit of information that describes a two-dimensional quantum system [49]. An important characteristic of quantum computation is that the qubit can be in a state of superposition, that is, the qubit can be in the $|0\rangle$

and $|1\rangle$ state simultaneously. Mathematically this is represented as a matrix of complex numbers:

$$|\psi\rangle = a|0\rangle + b|1\rangle \equiv \begin{pmatrix} a \\ b \end{pmatrix}, \tag{2}$$

where $|a|^2+|b|^2= 1$. Thus $|a|^2$ and $|b|^2$ represent the probability of finding the qubit after being measured in the state $|0\rangle$ and $|1\rangle$ respectively [49, 28].

Dirac's notation allows describing the state of a quantum system formally. For each "ket" $|\psi\rangle$ there is a corresponding "bra" $\langle\psi|$. The ket and the bra contain equivalent information about the quantum state. Mathematically, they are dual with each other, that is:

$$\langle\psi| = a^*\langle 0| + b^*\langle 1| = (a^*b^*). \tag{3}$$

Just as a single qubit can be found in superposition of the possible states $|0\rangle$ and $|1\rangle$, a register of n-qubits can be found in superposition of all 2^n possible states $|00\ldots0\rangle, |00\ldots1\rangle, \ldots, |11\ldots1\rangle$. For example the general form of the state of a 2 qubit quantum memory register is represented as:

$$|\psi\rangle = c_0|00\rangle + c_1|01\rangle + c_2|10\rangle + c_3|11\rangle, \tag{4}$$

where $|c_0|^2+|c_1|^2+|c_2|^2+|c_3|^2= 1$. This implies that we can have a register that contains many different bit strings, each with its corresponding amplitude value.

The general form of a n-qubit quantum memory register is:

$$\begin{aligned} |\psi\rangle = c_0|00\ldots0\rangle + c_1|00\ldots1\rangle + \ldots \\ + c_{2^n-1}|11\ldots1\rangle = \sum_{i=0}^{2^n-1} c_i|i\rangle, \end{aligned} \tag{5}$$

where $\sum_{i=0}^{2^n-1}|c_i|^2= 1$ and $|i\rangle$ represents the eigen state of the computational base whose bit values match those of the decimal number expressed in base 2 notation, padded on the left (if necessary) with bits "0" to make a full complement of n bits.

3.3 Quantum Inspired Algorithms

The concepts and principles of quantum mechanics to develop more efficient evolutionary computing methods were introduced by Narayanan and Moore in 1996 [33]. The main objective was to compare the performance of a classic algorithm and quantum-inspired algorithm in the Traveling Salesman Problem (TSP). They use an interference crossover operator, which consists of taking the first element of chromosome one, the second element of chromosome two, the third element of chromosome three, and so on; if an element already exists on the chromosome, another element that does not exist on the chromosome is chosen. The results showed that the quantum-inspired genetic algorithm outperformed the classical version.

In [32], a basic methodological principle to design a quantum algorithm was presented. The main objective was to identify the novelty and potential of the quantum algorithm in tackling NP-hard problems.

The two first pioneer works on quantum computing are Genetic Quantum Algorithm (GQA) proposed by [15] and Quantum Inspired Evolutionary Algorithm (QEA) introduced in [16]. GQA is based on quantum computing concepts and principles such as qubits and superposition instead of binary, numeric, or symbolic representation. GQA [15] demonstrated its effectiveness and applicability by experimental results on the 0-1 Knapsack Problem. All the experiments were simulated only in classic computers.

Quantum genetic algorithms have an excellent ability to perform global searches due to thier diversity in the population caused by the probabilistic representation; this characteristic allows for finding better solutions in a shorter time than the classical algorithms [17]. For example, with a single quantum register of three qubits, it is possible to represent eight states $(2^3 = 8)$; to represent eight states with a classical algorithm, eight registers would be needed. For large instances of N (large problems), a quantum computer requires only a 32-qubit register to handle all the possible combinations of 32-bit numbers, whereas a classical computer

requires 4,294,967,296 memory registers, which is significantly large.

The QEA [16] is the upgrade of the GQA [15], like the GQA, the QEA was designed to solve the 0-1 Knapsack Problem. The process to find the optimal solution in both algorithms is similar. The main difference between both proposals is the concept of migration introduced by QEA, which is a process that can induce a variation of the probabilities of a quantum chromosome [31].

The interaction between quantum computing and evolutionary computation can be addressed in three different ways [50]: The first is the Evolutionary-Designed Quantum Algorithm (EDQAs); here, the idea is to use genetic programming to generate new quantum algorithms. The second way is to use Quantum Evolutionary Algorithms (QEAs), focusing on developing evolutionary algorithms for quantum computers. The third way is Quantum-Inspired Evolutionary Algorithms (QIEAs), which use quantum mechanics concepts such as qubits, superposition, quantum gates, and quantum measurements to develop evolutionary methods for classic computers; a novel proposal in this category is the Quantum Inspired Acromyrmex Evolutionary Algorithm (QIAEA) [30].

Nowadays, companies such as D-wave [23], Google [3], and IBM [37] have created quantum computers with the ability to solve some problems that today's classical computers cannot. With the arrival of quantum supremacy (a term coined by John Preskill [39]) announced by Google in 2019 [3], it was mentioned that any problem that a classical computer could not solve in polynomial time could be solved by a quantum computer. However, In [40] mentioned that we are currently in the NISQ (Noisy Intermediate-Scale Quantum) era. *Intermediate scale* refers to the size of quantum computers available at present, and *Noisy* emphasizes that we will have imperfect control over those qubits. At present, the hardware for controlling trapped ions [4], or superconducting circuits [5], the error rate per gate for two-qubit gates is above the 0.1% [40], which means that with a sequence of 1000 quantum gate operations, the error rate in the circuit would be 100%, which would not give reliable results at all.

3.4 IBMQ Framework

The IBMQ framework is a software development kit (SDK) for performing quantum computations that utilize quantum mechanical principles such as superposition and entanglement. It allows the development of hybrid quantum computing algorithms that is an essential issue for our proposal. All the experiments were designed and run on the Qiskit platform, an open-source framework computational platform for working with quantum computers at the level of circuits, pulses, and algorithms. Qiskit is made up of four fundamental elements [19]. They are: 1) **Terra** that provides a base for composing quantum programs at the circuit level and pulses; with this module, we can perform optimizations for the constraints of a specific device. 2) **Aer** that allows accelerating the development of applications via simulators and noise models. 3) **Ignis** dedicated to fighting noise and errors; it is meant for those who want to work designing quantum error correction codes. 4) **Aqua** is where algorithms for quantum computing are built; this module focuses on constructing solutions for real-world application problems.

3.5 Hybrid Quantum Genetic Algorithm for solving the Knapsack Problem in the IBM-Q quantum computer

This section will describe our proposal to solve the knapsack problem using a quantum hybrid genetic algorithm. Hybrid computation in the NISQ era is an good way to perform quantum computing to no loose coherency, especially when the length of quantum circuits is high, which is the case of quantum meta-heuristics.

Figure 1 shows a block diagram where a quantum hybrid genetic algorithm is depicted. On the right side are the computation steps that are executed by the classical computer. On the left side are the steps that are performed by the quantum computer. As was mentioned, this model of computation will help us to maintain quantum circuits of small lengths. To understand better this figure is convenient to use Algorithm 1 and Algorithm 2. The computational steps that are achieved by the quantum computer are 3, 5, and 8 of Algorithm 1; the CC executes the remaining.

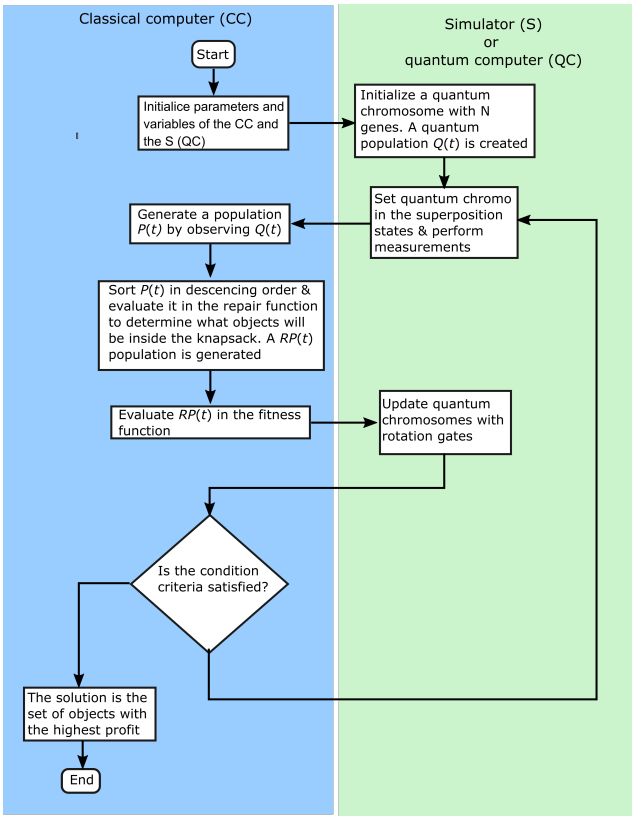

Fig. 1. Hybrid genetic algorithm. On the right side are the classical computer's steps. On the left side, the steps executed by the quantum computer or quantum simulator. In the figure, RP is the classic population after having been evaluated in the repair function

The CC performs the HQGA's parameter initialization for both systems, the classical and the QC. In this first stage, the quantum chromosome length is defined, and this information is sent to the QC. In the QC, the quantum chromosome $|\Psi_i^t\rangle$ is set in the superposition state where a number n of measurements is achieved to create an initial classical population P. In the CC, this population is repaired, evaluated, and sorted in descending order.

Algorithm 1 describes our algorithmic implementation proposal of the Hybrid Quantum Genetic Algorithm (HQGA) to solve the knapsack problem. The input of the HQGA is a set of quantum chromosomes; i.e, the initial quantum population is defined as $Q(t) = [\Psi_1^t, \Psi_2^t, \ldots, \Psi_n^t]$, where n is

Algorithm 1 Hybrid Quantum Genetic Algorithm

Input: The number n of quantum chromosomes, and the maximal number of generations MAX_GEN

Output: The best solution b

1: **Begin**
2: $t \longleftarrow 0$;
3: Initialize $Q(t)$
4: **while** $t < MAX\_GEN$ **do**
5: Measure $Q(t)$ to generate $P(t)$
6: Repair $P(t)$
7: Evaluate $RP(t)$
8: Update $Q(t)$
9: Store the best solution b of $P(t)$ in $B(t)$
10: **End**

the size of the population and t is the generation number. A quantum chromosome is defined as follows:

$$|\Psi_i^t\rangle = \left[\begin{array}{c|c|c|c} \alpha_{i,1}^t & \alpha_{i,2}^t & \cdots & \alpha_{i,m}^t \\ \beta_{i,1}^t & \beta_{i,2}^t & \cdots & \beta_{i,m}^t \end{array} \right], \qquad (6)$$

where m is the number of qubits and $i = 1, 2, \ldots, n$. The length of a qubit string is the same as the number of items.

The algorithm's output will be the best classical solution through generation saved in $B(t)$. In a similar fashion to any evolutionary algorithm, we started by using a generation counter t.

In step 2, the generation counter is initialized. In Step 3, we initialized the quantum chromosomes $Q(t)$ in the zero states, and then, they were put in the superposition state using the Hadamard gate; i.e.; for each quantum chromosome, we performed the next quantum operation $|\Psi_i\rangle|\otimes|H^{\otimes n}\rangle$. For the experiments, a population of one quantum chromosome was used

Step 4 is a while loop that will end when the maximum number of generations MAX_GEN has been reached. In step 5, the quantum population is observed (measured) to generate classical population (0 and 1) according to $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$, where α represents the probability that the qubit collapses to the zero state ($|0\rangle$) and β the probability that the qubit collapses to state one ($|1\rangle$) after being measured.

Algorithm 2 Repair Algorithm

Input: Classic chromosomes $P(t) = 0 \ldots 000 + \ldots + 1 \ldots 111_{2^n-1}$, n is the number of qubits
Output: $RP(t)$

1: **Begin**
2: knapsack-full $\leftarrow$ false
3: **if** $\sum_{j=1}^{m} w_j x_j > V$ **then** knapsack-overfilled $\leftarrow$ true
4: **while** knapsack-full = true **do**
5: Select a j-th item from the knapsack
6: $x_j \leftarrow 0$
7: **if** $\sum_{j=1}^{m} w_j x_j < V$ **then** knapsack-full $\leftarrow$ false
8: **while** knapsack-full = false **do**
9: Select a j-th item from the knapsack
10: $x_j \leftarrow 1$
11: **if** $\sum_{j=1}^{m} w_j x_j > V$ **then** knapsack-full $\leftarrow$ true
12: **End**

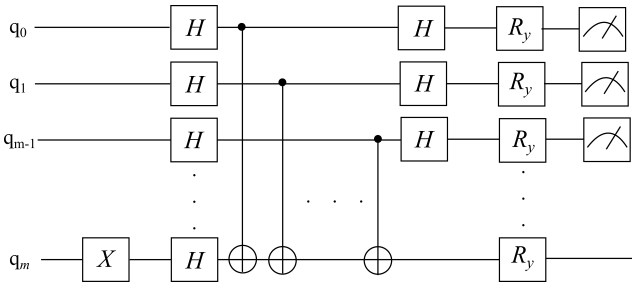

Fig. 2. Quantum circuit of the HQGA implemented in the IBM Qiskit simulator to solve the knapsack problem for 100, 250 and 500 objects

A repair algorithm was used in step 6, the main objective of this algorithm is to add or remove items from the knapsack as shown below.

The inputs of the Algorithm 2 are classic chromosomes $P(t)$ and the output will be classic repair chromosomes $RP(t)$. Algorithm 2 begins with the variable knapsack-full; this indicates if the knapsack is full (true) or not (false). If the sum of the weight w_j of each object x_j is greater than the total capacity V of the knapsack, it means that knapsack-full = true, and the algorithm continues in step 4; otherwise, it continues in step 8.

Continuing with step 7 of Algorithm 1 the profit of a solution x is evaluated by $\sum_{i=1}^{n} p_i x_i$, and it is used to find the best solution b to store in $B(t)$ (step 9) after the update of $|\Psi_i\rangle$, $i = 1, 2, \ldots, n$. A qubit chromosome $|\Psi_i\rangle$ is update (step 8) by using a $R_y(\theta)$ rotation gate. The j-th qubit value (α_j, β_j) is update as:

$$\begin{bmatrix} \alpha'_j \\ \beta'_j \end{bmatrix} = \begin{bmatrix} \cos(\theta_j) & -\sin(\theta_j) \\ \sin(\theta_j) & \cos(\theta_j) \end{bmatrix} \begin{bmatrix} \alpha_j \\ \beta_j \end{bmatrix}. \quad (7)$$

4 Experiments and Results

Figure 2 shows the quantum circuit of the HQGA in the Qiskit simulator (ibm_qasm_simulator) to solve the Knapsack Problem with 100, 250, and 500 objects. The ibm_qasm_simulator is a quantum environment designed by IBM; this simulator allows to generate circuits with a maximum of 32 qubits, which could be considered limited for the problem to be addressed. The simulator allows adding bits to the quantum register so that it is possible to use bits and qubits within the same register.

Quantum algorithms have shown that with a single evaluation of the oracle, it is possible to determine the state of the function [9, 21, 36]. The same principle can be used when solving the Knapsack Problem since with a single evaluation in the fitness function is possible to determine the profit of the 2^n possible configurations.

The circuit (see Figure 2) is composed of a Pauli-X gate (X) which serves as an ancillae gate that will enable the C-NOT (Controlled NOT) gates to be activated; these generate entanglement between each of the qubits. Hadamard gates (H) are also shown, whose main objective is to place the qubits in superposition, as well as rotation gates $(R_y(\theta))$ which are used to update the quantum population.

According to [47] one of the main characteristics to indicate the complexity of a quantum circuit is its length (number of serial gate operations after having parallelized the circuit to the maximum extent possible). The quantum circuit's length is 5 and the width (total number of qubits on which the circuit acts, including any ancillae qubit) is 21 qubits.

In [15, 17, 16] it is mentioned that the number of bits (qubits, in our case) must be equal to the number of objects in the knapsack.

Table 1. Experimental results of the knapsack problem with hybrid quantum genetic algorithm

N	Best	Worst	Average	Std
100	666.23	662.60	663.89	0.945
250	1591.03	1591.010	1591.01	0.003
500	3347.33	3340.91	3347.12	1.173

Table 2. Experimental results of hybrid quantum genetic algorithm with 100 objects

Run	Generations	Rotation angles	Best
1	125	2.0420	663.78
2	93	1.7122	663.78
3	113	2.1049	663.50
4	62	1.4608	663.85
5	93	1.7122	665.85
6	113	1.8378	663.58
7	104	1.9792	663.50
8	72	1.6650	663.85
9	117	2.2619	663.15
10	102	1.9792	663.50
11	144	2.1991	663.19
12	94	1.9792	663.13
13	122	2.2305	663.78
14	956	7.6498	663.85
15	105	2.0892	663.85
16	84	1.7436	663.19
17	234	2.6075	663.17
18	930	8.1524	663.58
19	120	1.9792	663.58
20	106	1.8221	665.85
21	120	2.0420	663.50
22	72	1.5708	665.54
23	203	2.3876	663.42
24	**97**	**1.7907**	**666.23**
25	125	2.1677	663.78
26	119	2.1677	663.17
27	121	2.1677	663.78
28	132	2.1520	665.85
29	123	1.9321	663.42
30	121	1.9949	662.60

However, the Qiskit simulator only allows a maximum of 32 qubits, but to not overload the system's RAM memory it was decided to use only 21 qubits, to complete the remaining qubits, bits were added to the quantum register.

In all the experiments, the following data sets were considered:

$$\begin{aligned} w_i &= uniformly\, random[1, 10) \\ p_i &= w_i + 5, \end{aligned} \tag{8}$$

and the average knapsack capacity was calculated as $V = 1/2 \sum_{i=1}^{m} w_i$. One of the most accepted ways to classify the complexity of an instance is the correlation between its data: the relationship or not between the weights of each object and its profit [29, 38]. This complexity is defined as: Uncorrelated, Weakly correlated, Strongly correlated, Inverse strongly correlated, Almost strongly correlated, and Subset sum. Where strongly correlated instances are hard to solve [38].

We carried out experiments to test the algorithm's performance. We ran the algorithm 30 times during 1000 generations, with strongly correlated data sets of 100, 250, and 500 objects. the IBM Qiskit simulator was used.

Table 1 shows the experimental results of HQGA solving the Knapsack Problem with 100, 250, and 500 objects. The "N" column represents the number of objects in the knapsack, the "Best", "Worst", "Average", and "Std" columns, represent the best solution, the worst solution, the average solution, and the standard deviation, respectively.

The experimental results demonstrated the robustness of the HQGA solving the Knapsack Problem with different quantities of objects because the standard deviation (Std) presented in Table 1 shows small values, even very close to zero.

Tables 2, 3, and 4 show the experiments with 100, 250, and 500 objects, respectively. For each table, the "Run" column represents the run's number, and the "Generations" column shows the generation where the best solution was obtained in each run; the "Rotation angles" column represents the angle of the rotation gate in that generation, the "Best" column shows the best solution obtained in each run.

Table 4 shows the experimental results with 500 objects. The best solution (bold letters) was in the first run and the worst solution (italic letters) in run 12. The rotation angles was 37/50π and 28/50π respectively.

Table 3. Experimental results of hybrid quantum genetic algorithm with 250 objects

Run	Generations	Rotation angles	Best
1	160	2.3562	1,591.01
2	122	2.0420	1,591.01
3	137	1.9635	1,591.01
4	134	2.1834	1,591.01
5	162	2.4347	1,591.01
6	122	1.9792	1,591.01
7	113	1.9321	1,591.01
8	126	2.0892	1,591.01
9	169	2.2934	1,591.01
10	146	2.2305	1,591.01
11	103	1.8692	1,591.01
12	137	2.1206	1,591.01
13	133	2.0263	1,591.01
14	129	2.2462	1,591.01
15	117	2.1206	1,591.01
16	143	2.3248	1,591.01
17	131	1.9792	1,591.01
18	121	1.9949	1,591.01
19	161	2.2619	1,591.01
20	121	1.9792	1,591.01
21	131	2.0577	1,591.01
22	209	2.6232	1,591.01
23	139	2.1834	1,591.01
24	200	2.3091	1,591.01
25	114	1.8064	1,591.01
26	124	2.1206	1,591.01
27	113	1.9321	1,591.01
28	**104**	**1.7593**	**1,591.03**
29	129	1.9792	1,591.01
30	153	1.9949	1,591.01

Table 4. Experimental results of hybrid quantum genetic algorithm with 500 objects

Run	Generations	Rotation angles	Best
1	**247**	**2.3248**	**3,347.33**
2	150	1.9792	3,347.33
3	191	1.9635	3,347.33
4	155	1.7593	3,347.33
5	161	1.9478	3,347.33
6	168	2.1991	3,347.33
7	163	2.0263	3,347.33
8	148	1.7279	3,347.33
9	165	1.9007	3,347.33
10	177	1.9792	3,347.33
11	149	1.8850	3,347.33
12	156	1.7750	3,340.91
13	225	2.1677	3,347.33
14	209	1.9635	3,347.33
15	257	2.1520	3,347.33
16	154	1.8692	3,347.33
17	161	1.9164	3,347.33
18	162	1.9792	3,347.33
19	225	2.0735	3,347.33
20	200	2.1049	3,347.33
21	227	2.2305	3,347.33
22	182	1.9792	3,347.33
23	174	1.9949	3,347.33
24	177	2.1991	3,347.33
25	159	2.0420	3,347.33
26	164	1.8064	3,347.33
27	244	2.3405	3,347.33
28	257	2.3405	3,347.33
29	190	2.3091	3,347.33
30	803	3.5657	3,347.33

In Table 2, the best solution was obtained in run 24 (bold letters) with a rotation angle of 14/25π and the worst in run 30 (italic letters) with a rotation angle of 16/25π.

In some runs a big difference can been seen between the rotation angles (see Table 2). For example, in run 18, the rotation gate has an angle of 8.1524 rad (13/5π), and the run 17 has an angle of 2.6 rad (41/50π); this difference is due to the intrinsic probabilistic condition of quantum algorithms and the number of bits and qubits that the quantum register has. The last characteristic will be explained later.

In Table 3, the experimental results with 250 objects is presented. The worst solution (italic letters) was obtained in run 1 with a rotation angle of 15/20π and the best solution (bold letters) in run 28 with a rotation angle of 11/20π.

The best solutions presented by Table 3 and 4 show values very similar, unlike those presented by Table 2.

This behavior is due to the number of bits (qubits) that the quantum register handles and how the rotation angle is determined; as mentioned above, the number of objects in the knapsack is equal to the number of bits and qubits that the quantum register has. For example, for 250 objects, the quantum register stores 250 bits (the qubits have already been measured). To determined the rotation angle, the $i-$th bit of the

Table 5. θ values for the rotation gates

x_i	b_i	$f(x) \geq f(b)$	$\Delta\theta$
0	0	false	0
0	0	true	0
0	1	false	0
0	1	true	0.05
1	0	false	0.01
1	0	true	0.025
1	1	false	0.005
1	1	true	0.025

Table 6. Best, worst, average solutions, and standard deviation (Std) with 100 objects. Best results in bold

Algorithm	Best	Worst	Average	Std
HQGA	**666.23**	**662.60**	**663.89**	0.945
HQGA-Q	636.09	636.09	636.09	**2.31e-13**
QIEA	663.50	656.01	659.87	2.812
QIEA-Q	660.92	599.60	632.37	19.739

current quantum register and the $i-$th bit of the best solution obtained up to that moment are compared (see Table 5), as there are more bits in the register the probabilities to obtaining large rotation angle values are lower, causing the best solutions to remain constant.

The rotation angles for all experiments were set using Table 5 and obtained from [15, 17]. Where x_i and b_i represent the i-th bits for the binary solution x and the best solution b, respectively. The third column ($f(x) \geq f(b)$) is the comparison between the evaluation of the binary solution and the evaluation of the best solution. The column $\Delta\theta$ represents the rotation value for the rotation gate.

For testing the performance of the HQGA, it was compared against a modified version of HQGA (HQGA-Q), a Quantum Inspired Evolutionary Algorithm (QIEA), and a modified version of QIEA (QIEA-Q). The quantum circuits used for the aforementioned algorithms are the same as shown in Figure 2. They were all implements in the ibm_qasm_simulator. The operation of the HQGA-Q is very similar to the HQGA; the only difference is that the HQGA-Q uses only qubits in its quantum register, while the HQGA uses bits and qubits. For example, with 100 objects, the HQGA has in its quantum register 21 qubits and 79 bits (as we mentioned at the beginning of the section, the number of objects must be equal to the number of bits and qubits), and the HQGA-Q begins with a quantum register of 5 qubits, then the quantum register is measured 20 times, at the end a population of 100 bits is obtained. The same happens between the QIEA and QIEA-Q.

The experimental results of the HQGA, HQGA-Q, QIEA, and QIEA-Q with 100, 250 and 500 objects are presented in Tables 6, 7, and 8 respectively. All the experiments were carried out in the Qiskit IBM simulator with 30 runs and 1000 generations, except for QIEA with 250 and 500 objects; in both cases, 20 runs were carried out.

The results presented in Table 6 demonstrate the superiority of the HQGA over its quantum counterpart. However, the HQGA-Q shows a lower standard deviation (Std); this is due to the number of qubits that the algorithm uses. In all experiments, five qubits were used for the HQGA-Q and the QIEA-Q. With 5 qubits $2^5 = 32$ possible solutions can be obtained, while with 21 qubits (quantity used for the HQGA and QIEA), $2^{21} = 2,097,152$ possible solutions are obtained. This indicates a greater diversity in both algorithms that generate better results.

With 250 objects (see Table 7), the HQGA outperform the rest of the algorithms (HQGA-Q, QIEA, and QIEA-Q) in all cases, presenting even a lower standard deviation, demonstrating its stability and better performance.

Table 8 shows the results with 500 objects, in this case we can see that HQGA and QIEA have the same best values, but QIEA has better standard deviation (Std), worst, and average solutions. In all the experiments HQGA and QIEA have better results than HQGA-Q and QIEA-Q.

By observing the three Tables (6, 7, and 8), we can see that the best solutions are provided by the HQGA and QIEA, as it was mentioned, both algorithms use bits and qubits in their quantum register, instead of HQGA-Q and QIEA-Q, whose algorithms use only qubits.

This shows that the use of bits and qubits in algorithms generates greater diversity allowing them to find better solutions in the search space.

Figure 3 shows the distribution in box plot of the best solutions for each of the quantum-inspired

Table 7. Best, worst, average solutions, and standard deviation (Std) with 250 objects. Best results in bold

Algorithm	Best	Worst	Average	Std
HQGA	**1591.03**	**1591.01**	**1591.01**	**0.003**
HQGA-Q	1576.2	1553.86	1562.37	5.701
QIEA	1591	1590.87	1591	0.031
QIEA-Q	1568.2	1548.08	1559.92	6.051

Table 8. Best, worst, average solutions, and standard deviation (Std) with 500 objects. Best results in bold

Algorithm	Best	Worst	Average	Std
HQGA	**3347.33**	3340.91	3347.12	1.173
HQGA-Q	3260.34	3169.92	3218.42	31.14
QIEA	3347.33	**3347.33**	**3347.33**	**1.86e-12**
QIEA-Q	3257.9	3158.16	3222.64	34.241

methodologies with 100 objects. The HQGA has a higher median demonstrating that it had greater number of better solutions than the HQGA-Q, QIEA, and QIEA-Q. Also, the range is very short comparing with the other methodologies, except for the HQGA-Q, where the range is almost zero. We can see that with a few objects, the HQGA-Q has a robust performance.

Figure 4 shows the best solutions for the knapsack problem with 250 objects in a box plot. Here, we can see a similar behavior between the algorithms that combined bits and qubits (HQGA and QIEA). Also, the algorithms that only use qubits (HQGA-Q and QIEA-Q) have almost the same behavior between each other, since the distribution of their data and the median is very similar for each pair of algorithms.

Another point worth highlighting is that the HQGA-Q presents better distribution in its solutions, unlike those shown in Figure 3, while QIEA-Q shows a lower distribution in its solutions. This demonstrates how the number of objects affects both algorithms' performance as the number of objects increases, while the HQGA and QIEA do not show this behavior.

The behavior of the HQGA and the QIEA shown in Figure 5 is very similar to that presented in Figure 4, even to that shown in Figure 3. This demonstrates the stability of both algorithms when

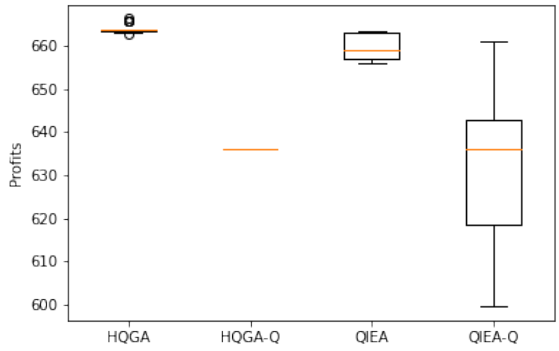

Fig. 3. Box plot of the best profits (solutions) of the HQGA, HQGA-Q, QIEA, QIEA-Q with 100 objects

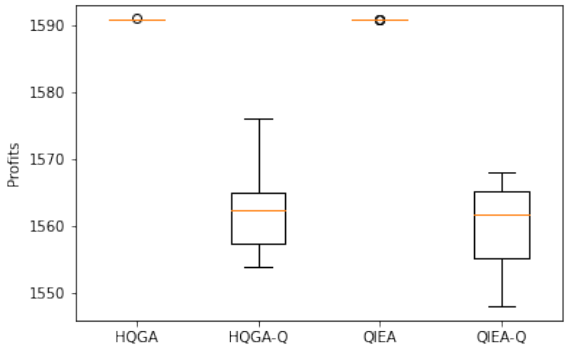

Fig. 4. Box plot of the best profits (solutions) of the HQGA, HQGA-Q, QIEA, QIEA-Q with 250 objects

solving the knapsack problem with a different number of objects.

On the other hand, the HQGA-Q and the QIEA-Q present behaviors very different from those shown in Figures 3 and 4, with a negative asymmetric distribution where most of their data tend towards the third quartile demonstrating the instability of the algorithms that only use qubits (HQGA-Q and QIEA-Q) in solving the knapsack problem.

From the previous results, we can see that the HQGA and QIEA proposals are the ones that have shown the best results when solving the knapsack problem with 100, 250, and 500 objects. For this reason, Figure 6, 7, and 8 present the performance

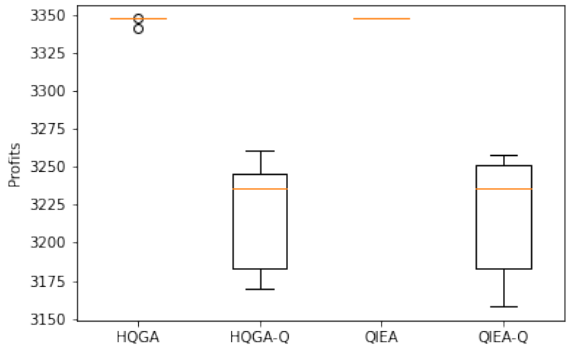

Fig. 5. Box plot of the best profits (solutions) of the HQGA, HQGA-Q, QIEA, QIEA-Q with 500 objects

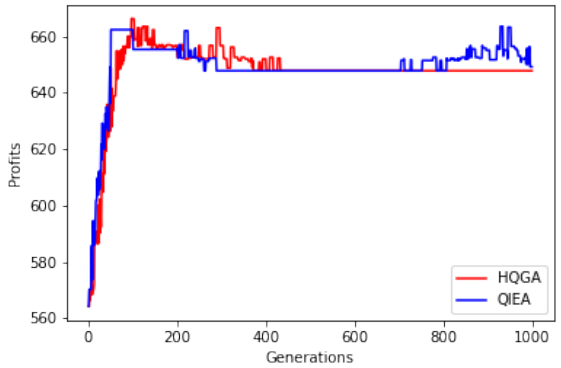

Fig. 6. Performance of the best run of HQGA vs QIEA with 100 objects

of both algorithms in solving the knapsack problem with 100, 250, and 500 objects, respectively.

From Figure 6 a comparison between the performance of HQGA and QIEA solving the knapsack problem with 100 objects is obtained. The solid red line represents the best run of the HQGA, and the solid blue line represents the performance of the best run of the QIEA. The runs number 24 and 6 were the best for the HQGA and QIEA, respectively. For HQGA, a stable solution is reached after generation number 400, while for the QIEA, a stable solution is never reached. The best solution of the HQGA was obtained in generation number 97. For the QIEA, the best solution occurred in generation number 92. It can be seen that the HQGA finds a better solution to the problem in fewer generations than the QIEA, in addition to presenting constant solutions after a certain number of generations, which shows the stability of the HQGA.

With 250 objects, the HQGA in Figure 7 shows its best performance in run 28. The best solutions were obtained in generation 104, and the algorithm reached stable solutions after generation 400 (specifically generation 463). The QIEA had its best performance in the first run, reaching the best solution in generation 101 and the stability after generation 300. For this problem, it can be seen that both algorithms reached stable solutions after a certain number of generations, unlike what is shown in Figure 6, where the QIEA cannot achieve this behavior, we can even see how the migration operator in the QIEA allows obtaining constant solutions through certain generations. However, this does not mean that the QIEA is more stable than the HQGA since with 100 objects (Figure 6) it could be seen that the HQGA was the one that found stable solutions, while QIEA did not present the same behavior.

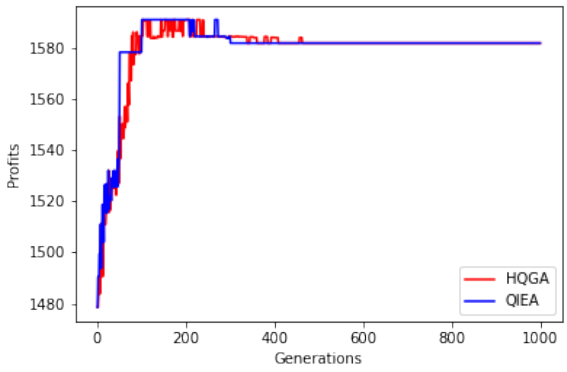

Fig. 7. Performance of the best run of HQGA vs QIEA with 250 objects

The performance of the HQGA and QIEA solving the knapsack problem with 500 objects (see Figure 8) is very similar to that presented in Figure 7. In both cases, the first run was the best. The HQGA obtained its best solution at generation 247

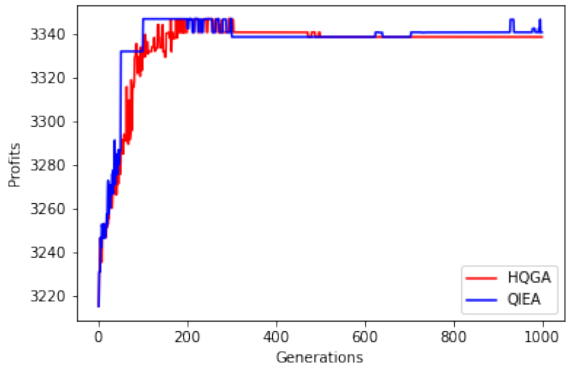

Fig. 8. Performance of the best run of HQGA vs QIEA with 500 objects

and stable solutions after generation 300. QIEA reached its best solution at generation 205 and stable solutions after generation 300, but we can see a small peak at generation 600 and after generation 900. While the HQGA only shows constant solutions.

With 250 and 500 objects (Figure 7 and 5) the QIEA obtained a best solution before the HQGA showing a better convergence time. Nevertheless, the HQGA in all the experiments (100, 250 and 500 objects) has shown a better stability and better optimal solutions.

So far only experiments comparing hybrid quantum algorithms have been shown. However, it is necessary to compare the performance of the HQGA with a classical genetic algorithm (GA) to get a complete picture of the HQGA's behavior. GA was run 30 times, with 1000 generations on three strongly correlated data sets (100, 250, and 500 objects).

Table 9. HQGA and GA experimental results with 100 objects. Best results in bold

Algorithm	Best	Worst	Average	Std
HQGA	666.23	**662.6**	663.89	**0.945**
GA	**712.18**	658.03	**687.35**	12.39

For the selection process, the tournament selection operator was used, in the recombination process the single one-point crossover operator was used, and finally a mutation operator. In all the experiments a population of 100 chromosomes, a recombination rate of 0.8, and a mutation rate of 0.4 were used.

Table 10. HQGA and GA experimental results with 250 objects. Best results in bold

Algorithm	Best	Worst	Average	Std
HQGA	1591.03	1591.01	1591.01	**0.003**
GA	**1688.42**	**1608**	**1651.16**	16.61

Table 11. HQGA and GA experimental results with 500 objects. Best results in bold

Algorithm	Best	Worst	Average	Std
HQGA	3347.33	3340.91	3347.12	**1.173**
GA	**3581**	**3513.86**	**3548.75**	19.6

Tables 9, 10, and 11 show the experimental results of the HQGA and GA with 100, 250, and 500 objects, respectively. In all the experiments, HQGA is outperformed by GA in obtaining the best solutions. Let us remember that both algorithms were executed in a classical computer, this means that the GA has the advantage of being able to generate its complete population of bits, while the HQGA being a hybrid algorithm and due to the limitations of the NISQ era mentioned above, it cannot generate the entire population of qubits, which limits the HQGA's ability to perform better. To correctly approach the comparison of both algorithms, it would be necessary to implement the HQGA in a quantum computer that allows the use of the necessary number of qubits.

Although, the HQGA was clearly surpassed by the GA in the best solutions, the HQGA obtained in all the experiments the smallest standard deviation (Std), demonstrating a better robustness even when implemented in a classical computer.

To see more clearly the performance of the HQGA and GA, in Figures 9, 10, and 11 the performance graphs of both algorithms are presented with 100, 250, and 500 objects, respectively. With 100 objects, Figure 9 clearly shows from the beginning of the generations the superiority of GA compared to HQGA.

However, the search capacity of the HQGA is better, as its first solution was approximately 370 and its best solution was 666.23, therefore there was a 1.8-fold increase over the initial solution. While GA's first solution was approximately 620, and its best solution was 712.18, the increase was 1.14 times, less than presented by the HQGA.

Figure 10 shows the performance of the HQGA and GA with 250 objects. In addition to the evident superiority of the GA over the HQGA, it can be noted that both proposals reached their best solution in the same number of generations (approximately in the 100th generation).

With 500 objects (Figure 11), things remain very similar to what was previously presented.

One of the main disadvantages of the HQGA is the execution time. Because the number of qubits (and bits) in a quantum register equals the number of objects in the knapsack, it is currently impossible to deal with problems with more than 15 elements [29].

For instance, a system with n-qubits can represent 2^n states simultaneously, which becomes a difficult task for a current classic computer. The tests were performed on personal computer equipment with 8 Gigabytes of RAM memory and a Intel(R) Core(TM) i7-3537U CPU @ 2.50 GHz processor.

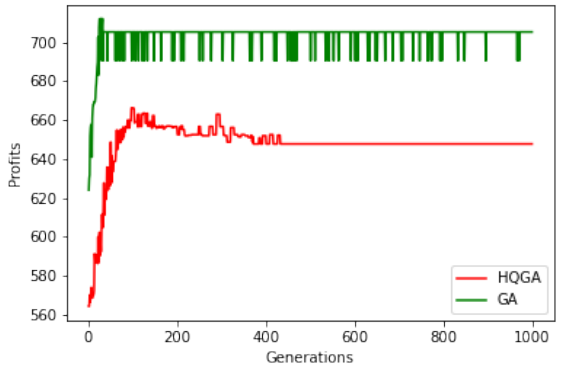

Fig. 9. Performance of the best run of HQGA vs GA with 100 objects

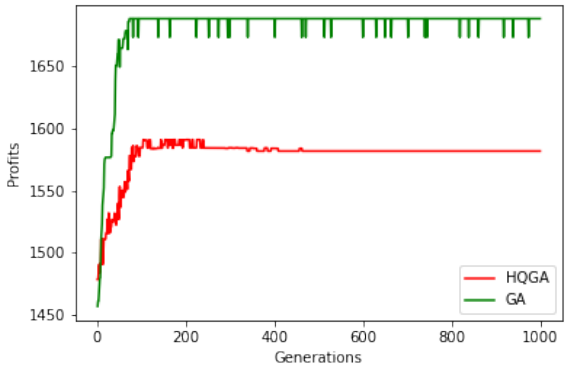

Fig. 10. Performance of the best run of HQGA vs GA with 250 objects

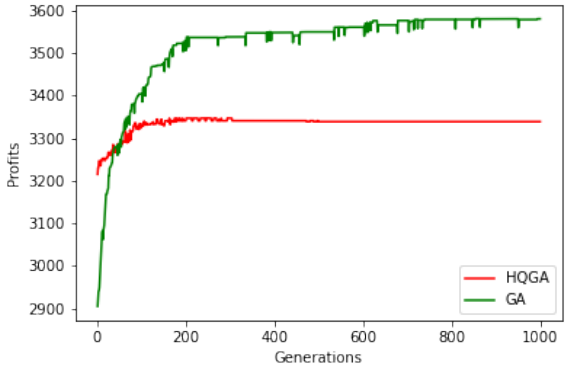

Fig. 11. Performance of the best run of HQGA vs GA with 500 objects

5 Conclusion and Future Work

This paper proposes a Hybrid Quantum Genetic Algorithm (HQGA), a modified version of the HQGA (HQGA-Q), a Quantum Inspired Evolutionary Algorithm (QIEA), and a modified version of QIEA (QIEA-Q) implemented in a quantum simulator to solve the 0-1 knapsack problem.

The novelty of this work is the design of the aforementioned quantum algorithms and their implementation in the Qiskit IBM quantum simulator. There are already some works in the state of the art that have implemented algorithms in the Qiskit IBM simulator [29, 44], solving the binary

knapsack problem and the traveling salesman problem, respectively. However, this proposal implemented and compared the performance of four quantum algorithms solving the knapsack problem with different number of objects using a strongly correlated data set. In addition, a quantum circuit was designed and implemented to generate the initial quantum population. This proposal until today has not been seen in any other work present in the state-of-the-art.

Also, the idea of incorporating bits and qubits to the designed algorithms was proposed and implemented, with the aim of solving the limitation of the quantum simulator.

The results showed that HQGA presents better solutions solving the knapsack problem with 100, 250, and 500 objects than the other hybrid quantum algorithms, except with GA, in this case HQGA was exceeded in all experiments. With hybrid quantum algorithm, HQGA presented a lower standard deviation in 1/3 (33%) of the cases. Furthermore, the HQGA in the performance test demonstrated a better stability in all the experiments than the QIEA. Although, in 2/3 of the experiments the QIEA reached the best solution before the HQGA showing a better convergence time. In none of the experiments, the QIEA-Q and HQGA-Q had better solutions than HQGA and QIEA.

The main disadvantage of the HQGA is the execution time and the ability to find no better solutions than Genetic algorithm (GA), as explained in section 4, the number of qubits (and bits) using by a quantum register in a current classic computer is limited, since for a n-qubit register 2^n states can be represented at the same time, to calculated that amount of data in a classic computer would be a difficult task. It must be remembered that the experiments were carried out on a classical computer using a simulated quantum environment (ibmq_qasm_simulator), and not on a real quantum computer where the natural parallelism of these computers would demonstrate their superiority.

For future work the HQGA, HQGA-Q, QIEA, and QIEA-Q will be implemented in a real IBM quantum computer and the results will be compared against the results presented in this work. Other methods will be tried to update the rotation angle for the quantum gates. The four quantum algorithms will be adapted to solve other kinds of combinatorial optimization problems, such as, the traveling salesman problem or path planning problems. Finally, the quantum-inspired algorithms proposed in this work will be modified to solve multi-objective combinatorial optimization problems.

Acknowledgments

We thank Instituto Politécnico Nacional (IPN), the Comisión de Fomento y Apoyo Académico del IPN (COFAA), and the Mexican National Council of Science and Technology (CONACYT) for supporting our research activities.

References

1. **Adeyemo, H., Ahmed, M. (2017).** Solving 0/1 knapsack problem using metaheuristic techniques. 9th IEEE-GCC Conference and Exhibition (GCCCE), pp. 1–6.
2. **Ajagekar, A., You, F. (2019).** Quantum computing for energy systems optimization: Challenges and opportunities. Energy, Vol. 179, pp. 76–89.
3. **Arute, F., Arya, K., Babbush, R., others (2019).** Quantum supremacy using a programmable superconducting processor. Nature, Vol. 574, pp. 505–510.
4. **Ballance, C., Harty, T., Linke, N., Sepiol, M., Lucas, D. (2016).** High-fidelity quantum logic gates using trapped-ion hyperfine qubits. Physical Review Letters, Vol. 117, No. 6.
5. **Barends, R., Kelly, J., Megrant, A., Veitia, A., Sank, D., Jeffrey, E., White, T. C., Mutus, J., Fowler, A. G., Campbell, B., et al. (2014).** Superconducting quantum circuits at the surface code threshold for fault tolerance. Nature, Vol. 508, No. 7497, pp. 500–503.
6. **Bhattacharjee, K. K., Sarmah, S. P. (2015).** A binary cuckoo search algorithm for knapsack problems. 2015 International Conference on Industrial Engineering and Operations Management (IEOM), pp. 1–5.

7. **Bhattacharjee, K. K., Sarmah, S. P. (2015).** A binary firefly algorithm for knapsack problems. 2015 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM), pp. 73–77.
8. **Braine, L., Egger, D. J., Glick, J., Woerner, S. (2021).** Quantum algorithms for mixed binary optimization applied to transaction settlement. IEEE Transactions on Quantum Engineering, Vol. 2, pp. 1–8.
9. **Cao, Z., Uhlmann, J., Liu, L. (2018).** Analysis of deutsch-jozsa quantum algorithm. IACR Cryptology ePrint Archive, Vol. 2018, pp. 249.
10. **Cook, S. A. (1971).** The complexity of theorem-proving procedures. IN STOC, ACM, pp. 151–158.
11. **Du, K.-L., Swamy, M. (2016).** Search and Optimization by Metaheuristics.
12. **Forno, E., Acquaviva, A., Kobayashi, Y., Macii, E., Urgese, G. (2018).** A parallel hardware architecture for quantum annealing algorithm acceleration. 2018 IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC), pp. 31–36.
13. **Gao, Y., Zhang, F., Zhao, Y., Li, C. (2018).** Quantum-inspired wolf pack algorithm to solve the 0-1 knapsack problem. Mathematical Problems in Engineering, Vol. 2018, pp. 1–10.
14. **García, J., Crawford, B., Soto, R., Castro, C., Paredes, F. (2018).** A k-means binarization framework applied to multidimensional knapsack problem. Applied Intelligence, Vol. 48, No. 2, pp. 357–380.
15. **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. CEC00 (Cat. No.00TH8512), volume 2, pp. 1354–1360.
16. **Han, K.-H., Kim, J.-H. (2003).** Quantum-inspired evolutionary algorithm for a class of combinatorial optimization. Evolutionary Computation, IEEE Transactions on, Vol. 6, pp. 580–593.
17. **Han, K.-H., Park, K.-H., Lee, C.-H., Kim, J.-H. (2001).** Parallel quantum-inspired genetic algorithm for combinatorial optimization problem. Proceedings of the 2001 Congress on Evolutionary Computation (IEEE Cat. No.01TH8546), volume 2, pp. 1422–1429.
18. **Huang, Y., Wang, P., Li, J., Chen, X., Li, T. (2019).** A binary multi-scale quantum harmonic oscillator algorithm for 0–1 knapsack problem with genetic operator. IEEE Access, Vol. 7, pp. 137251–137265.
19. **IBM (2020).** The qiskit elements. https://quantum-computing.ibm.com/docs/qiskit/the_elements.
20. **Jindal, A., Bansal, S. (2019).** Effective methods for constraint handling in quantum inspired evolutionary algorithm for multi-dimensional 0–1 knapsack problem. 2019 4th International Conference on Information Systems and Computer Networks (ISCON), pp. 534–537.
21. **Johansson, N., Larsson, J.-A. (2017).** Efficient classical simulation of the Deutsch–Jozsa and Simon's algorithms. Quantum Information Processing, Vol. 16, No. 9.
22. **Khemakhem, M., Chebil, K. (2016).** A tree search based combination heuristic for the knapsack problem with setup. Computers & Industrial Engineering, Vol. 99, pp. 280–286.
23. **King, J., Yarkoni, S., Raymond, J., Ozfidan, I., King, A. D., Nevisi, M. M., Hilton, J. P., McGeoch, C. C. (2017).** Quantum annealing amid local ruggedness and global frustration.
24. **Lai, X., Hao, J., Yue, D., Gao, H. (2018).** A diversification-based quantum particle swarm optimization algorithm for the multidimensional knapsack problem. 2018 5th IEEE International Conference on Cloud Computing and Intelligence Systems (CCIS), pp. 315–319.
25. **Lai, X., Hao, J.-K., Fu, Z.-H., Yue, D. (2020).** Diversity-preserving quantum particle swarm optimization for the multidimensional knapsack problem. Expert Systems with Applications, Vol. 149, pp. 113310.
26. **Lai, X., Hao, J.-K., Yue, D. (2019).** Two-stage solution-based tabu search for the multidemand multidimensional knapsack problem. European Journal of Operational Research, Vol. 274, No. 1, pp. 35–48.
27. **Li, H. (2019).** An angle-expressed quantum evolutionary algorithm for quadratic knapsack problem. IOP Conference Series: Materials Science and Engineering, Vol. 631, pp. 052054.
28. **Liu, C.-L., Wan, M.-H., Yang, J.-Y. (2010).** An improved quantum genetic algorithm and its application. 2010 International Conference on Computer Application and System Modeling, pp. 413–418.
29. **López-Sandoval, D., Cobos, C. (2020).** Adiabatic quantum computing applied to the solution of the binary knapsack problem. RISTI - Revista Iberica

de Sistemas e Tecnologias de Informacao, Vol. 38, pp. 214–227.

30. **Montiel Ross, O., Rubio, Y., Olvera, C., Rivera, A. (2019).** Quantum-inspired acromyrmex evolutionary algorithm. Scientific Reports, Vol. 9.
31. **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.
32. **Narayanan, A. (1999).** Quantum computing for beginners. Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406), Vol. 3, pp. 2231–2238.
33. **Narayanan, A., Moore, M. (1996).** Quantum-inspired genetic algorithms.
34. **Nezamabadi-pour, H. (2015).** A quantum-inspired gravitational search algorithm for binary encoded optimization problems. Engineering Applications of Artificial Intelligence, Vol. 40, pp. 62–75.
35. **Ozsoydan, F. B., Baykasoglu, A. (2019).** A swarm intelligence-based algorithm for the set-union knapsack problem. Future Generation Computer Systems, Vol. 93, pp. 560–569.
36. **Paredes López, M., Meneses Viveros, A., Morales-Luna, G. (2018).** Algoritmo cuántico de Deutsch y Jozsa en GAMA. Revista mexicana de física, Vol. 64, pp. 181–189.
37. **Pednault, E., Gunnels, J. A., Nannicini, G., Horesh, L., Wisnieff, R. (2019).** Leveraging secondary storage to simulate deep 54-qubit sycamore circuits.
38. **Pisinger, D. (2005).** Where are the hard knapsack problems? Computers & Operations Research, Vol. 32, No. 9, pp. 2271–2284.
39. **Preskill, J. (2012).** Quantum computing and the entanglement frontier.
40. **Preskill, J. (2018).** Quantum computing in the NISQ era and beyond. Quantum, Vol. 2, pp. 79.
41. **Sapra, D., Sharma, R., Agarwal, A. P. (2017).** Comparative study of metaheuristic algorithms using knapsack problem. 7th International Conference on Cloud Computing, Data Science Engineering - Confluence, pp. 134–137.
42. **Shi, H. (2006).** Solution to 0/1 knapsack problem based on improved ant colony algorithm. IEEE International Conference on Information Acquisition, pp. 1062–1066.
43. **Si, L., Shi, L., Wang, Y. (2010).** A novel self-organizing quantum evolutionary algorithm for multi-objective optimization. 2010 International Conference on Educational and Network Technology, pp. 499–503.
44. **Srinivasan, K., Satyajit, S., Behera, B. K., Panigrahi, P. K. (2018).** Efficient quantum algorithm for solving travelling salesman problem: An IBM quantum experience.
45. **Tkachuk, V. (2018).** An adaptive quantum evolution algorithm for 0–1 knapsack problem. System research and information technologies, pp. 77–88.
46. **Tkachuk, V., Tkachuk, O. (2018).** Higher-order quantum genetic algorithm for 0-1 knapsack problem. System research and information technologies, pp. 52–67.
47. **Williams, C. (2011).** Explorations in Quantum Computing.
48. **Yang, S., Jiang, Y., Nguyen, T. T. (2012).** Metaheuristics for dynamic combinatorial optimization problems. IMA Journal of Management Mathematics, Vol. 24, No. 4, pp. 451–480.
49. **Yanofsky, N., Manucci, M. (2008).** Quantum computing for computer scientists.
50. **Zhang, G. (2011).** Quantum-inspired evolutionary algorithms: A survey and empirical study. J. Heuristics, Vol. 17, pp. 303–351.
51. **Zhang, R., Gao, H. (2007).** Improved quantum evolutionary algorithm for combinatorial optimization problem. 2007 International Conference on Machine Learning and Cybernetics, volume 6, pp. 3501–3505.
52. **Zhou, Y., Chen, X., Zhou, G. (2016).** An improved monkey algorithm for a 0-1 knapsack problem. Applied Soft Computing, Vol. 38, pp. 817–830.

Article received on 20/06/2021; accepted on 17/11/2021.
Corresponding author is Enrique Ballinas.