EasyChair Preprint

№ 6092

Implementable Simple Quantum Genetic Algorithm

Mikel Garcia de Andoin and Javier Echanobe

July 16, 2021

Implementable Simple Quantum Genetic Algorithm

1st Mikel Garcia de Andoin
Department of Electricity and Electronics
University of the Basque Country UPV/EHU
Bilbao, Spain
mikelgda@gmail.com

2nd Javier Echanobe
Department of Electricity and Electronics
University of the Basque Country UPV/EHU
Bilbao, Spain
franciscojavier.echanove@ehu.es

***Abstract*—Quantum machine learning (QML) is a relatively recent field of research in which the areas of Quantum Computing (QC) and Machine Learning (ML) are merged in different ways and at different levels. In this paper, we propose a quantum genetic algorithm (QGA) that is a direct translation of the classical genetic algorithm (GA). Compared to other existing works, our proposal allows a simpler and more direct implementation, and therefore with less hardware requirements. QGA is compared with its classical counterpart through a function optimization benchmark, showing that both algorithms are equivalent. The results suggest future work of exploring similar algorithms and strategies in the search of quantum advantages.**

***Index Terms*—Quantum Computing, Quantum Algorithms, Genetic Algorithms, Optimization.**

I. Introduction

The theoretical and experimental scientific developments of the last decades have made it possible to advance and consolidate the discipline known as quantum computing (QC) [1]. This paradigm takes advantage of the particular laws of the quantum world such as superposition and entanglement, to perform calculations more quickly and efficiently. Although at the moment the systems developed are on a very small scale and are almost always limited to laboratory systems, it has been shown that for certain types of problems such as number factoring or search problems, quantum computers would be infinitely faster than conventional computers. For this reason, there are currently numerous research groups working hard to make quantum computers a reality in the near future.

The paradigm known as machine learning (ML) [2], encompasses a series of techniques and algorithms aimed at providing computers with prediction, generalization, and decision-making capabilities based on knowledge extracted from collections of reference data or learning data. This field has also experienced strong development in recent years due to the increasing capacity of computers as well as the availability of huge amounts of data to be processed. Among the most representative ML algorithms we can mention clustering, decision trees, support vector machines (SVM), Bayesian networks, neural networks or genetic algorithms. All these techniques have demonstrated their potential in many different applications, offering truly impressive results on numerous occasions. Today, ML techniques are at the heart of many so-called artificial intelligence systems. However, all these techniques require a large amount of hardware resources and have high computational complexity. Therefore, new techniques or methods to speed up its execution will be always of great value and interest.

In an effort to obtain joint benefit from ML and QC techniques, the area of research known as Quantum Machine Learning (QML) [3] has more recently emerged. This discipline aims to merge the two aforementioned paradigmes to obtain techniques that allow to address different problems more efficiently. It can be found in the literature different ways in which that merge or fusion can be carried out. For example, classical ML algorithms can be applied to address problems in the field of QM [4], [5]. Also, classical ML algorithms can be implemented in quantum computers to speed-up its execution. See for example [6] for a review of the field. The work proposed here, falls in this second type; that is, we propose a quantum genetic algorithm (QGA) that can be truly implemented in a quantum computer in a very simple and direct way. In Section II we will make a review of existing quantum genetic algorithms and explain how our proposal has in comparison, interesting advantages.

The rest of the paper is organised as follows. Section II describe some of the main existing works dealing with the QGA topic. Section III explain the proposed QGA. In Section IV we present some experiments that show the feasibility of the proposal. Finally Section V is devoted to concluding remarks.

II. Related Works

Quantum genetic algorithms can be divided into two main groups, depending on the strategy used to test the fitness of the solution. On the one hand, algorithms based on different versions of the Grover's algorithm [7], use an oracle (sometimes called black-box) to inquire the state about its fitness. These algorithms are based on quantum circuits; i.e., they are implemented by means of basic quantum logic gates (e.g. Hadamard, Toffoli, Fredkin, CNOT). Examples of recent work for this kind of algorithms can be found in [8]–[10]. In these algorithms, an oracle $\hat{Q}$ takes the quantum state $|x\rangle$ as an input, and without altering it, it deposit the fitness value in some ancilla qubits

$$\hat{Q}\,|x\rangle\,|0\rangle = |x\rangle\,|F(x)\rangle\,, \tag{1}$$

where $F(x)$ is expressed as its binary expansion, $F(x) = \sum_{k=0}^{\ell} 2^{\ell} F(x)_{\ell}$. This way, one could access the information about the evolution of the fitness function at each iteration

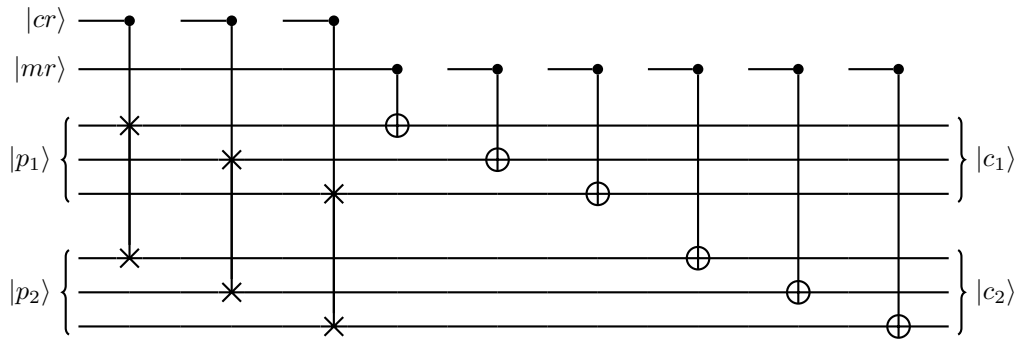

Fig. 1. Circuit for the simple QGA that implements the quantum crossover and mutation for a problem size n=3. Here, $p_{1,2}$ are the parents, and $c_{1,2}$ the children solutions. Each time the crossover or the mutation qubits are used, they are reset back to the state $|cr\rangle$ or $|mr\rangle$ respectively.

while not destroying the quantum state. At the end, the measurement of the state yields the solution to the problem. In general, constructing such oracle for a given fitness function is non trivial, and it takes up to $2^{n-1}(2^n-1)$ gates. That is, the complexity of the circuit (i.e. hardware resources) grows exponentially [11].

On the other hand, there are algorithms based on quantum annealing [12]. In this case, the problem is coded by means of a Hamiltonian and the solution is obtained once the ground state of the Hamiltonian is found. The system is initialized with a well known Hamiltonian and an easily preparable ground state. Then, the system is adiabatically evolved towards the problem Hamiltonian; i.e. maintaining the state of the system in the ground state at every time. At the end of the evolution, the state is measured, with high probability of being in the ground state of the system, thus, the solution of the problem. The main drawback of this algorithm is that the construction of a suitable Hamiltonian can be a non-trivial task. Sometimes it may not be implementable on an adiabatic computer or annealer (e.g. QWave) or it is required to apply separation techniques and implement small problems (e.g. 'qbsolv'). Moreover, the Hamiltonian can be somewhat funny, so that it is necessary to apply STA techniques or CD terms [13].

In this paper we propose a QGA, implemented as quantum circuit, that does not precise an oracle. In consequence, the computation scheme is much simpler and therefore, its implementation is easy and straightforward, requiring much less hardware resources. In the next section we carefully explain the details of this algorithm showing how it is able to converge to a solution despite not using an oracle.

III. Proposed Algorithm

Let us describe in this Section how is the Quantum Genetic algorithm proposed in this work.

The codification of the solutions into a quantum state in the computational basis is straightforward. A classical bit corresponding to 0(1) is coded in the ground(excited) state of a qubit $|0\rangle(|1\rangle)$. This way, any solution can be generated by starting from the ground state and applying a Pauli-x gate ($|0\rangle = X|1\rangle$, $|1\rangle = X|0\rangle$) to every qubit that has a 1 in the corresponding position.

Our algorithm is based on a simple Genetic Algorithm schema [14], [15]. It starts by generating m random solutions of size n. The algorithm evolves towards the optimal solution iteratively, generating new solutions until a stop condition is met. Each iteration can be divided into 3 main steps: parent selection, crossover and mutation.

Parent selection: At the start of each iteration 2 different parents are selected depending on its fitness. The algorithm favours the selection higher fitness solutions, so that the better individuals advance in the evolution process. For this task we can use various strategies, from which we decided to use the traditional Roulette Wheel selection. This strategy assigns to each individual a probability of being chosen (p_i), that is proportional to its relative fitness, $p_i = f_i / \sum_{j=1}^{m} f_j$.

Crossover: This step combines the bits of the parents into two new solutions. In the classical algorithm, first it is decided whether or not the crossover happens, depending on the crossover rate $cr \in [0,1]$. Then, using the single-point crossover, a random point on the string is selected. The two children will be the combination of the opposite halfs of the parents. This crossover can be applied in our algorithm just relabeling the qubits.

We propose to use a more quantum approach by implementing a quantum version of the uniform crossover. We can use Fredkin (or controlled SWAP) gates acting on the corresponding qubits of both parents (Fig. 1). The control qubit contains the information of cr, such that the qubits of the parents are flipped with probability cr, $|cr\rangle = \sqrt{1-cr}\,|0\rangle + \sqrt{cr}\,|1\rangle$. This state can be generated by applying a rotation on the y axis of an angle of $\theta_{cr} = 2\arcsin(\sqrt{cr})$, $|cr\rangle = R_y(\theta_{cr})\,|0\rangle$.

Mutation: In the mutation step, each bit of the children can flip with a probability $mr \in [0,1]$. This can be directly implemented in a quantum algorithm using CNOT gates. Similar to the previous step, this probability can be coded into a qubit that acts as the control of the CNOT gates acting

on the children's qubits.

The crossover and mutation steps are implemented using a quantum circuit (Fig. 1). At the end of each circuit run, the children qubits are measured in the z axis. Once the new solutions are converted to bits, we can compute and assign the fitness to each one.

A new generation is completed after a new population is full, that's it, after $m/2-1$ iterations. To prevent the best individuals from being lost, we use replacement-with-elitism. The 2 individuals with the best fitness value are automatically introduced in the next generation. This assures that the solution that the algorithm returns is the best solution found so far.

The GA produces new generations until it meets the stopping condition. The most naive condition is to preset a maximum number of iterations, $maxIter$. We can also set a certain number of iterations to let the GA to evolve, $converIter$. If after $converIter$ generations no new best solution is found, the algorithm finishes.

The No Free Lunch Theorem [16] states that we can not find the set of parameters that optimizes an algorithm for every possible problem. In other words, there is no universal optimization strategy. The input parameters used to run GA have to be chosen by intuition or by running similar problems, i.e. empirically.

The proposed algorithm (Alg. 1) can be run on quantum computers without the need to use quantum error correction. Flip-errors can be absorbed as an extra probability to the mutation rate. Gate errors, in both Fredkin and CNOT gates, reduce the effective cr and mr. Dephasing noise dones not interfere with our algorithm, as all the information is coded into the probabilities to measure the excited and the ground state. The only noise that could affect the outcome of the algorithm is the decoherence. Fortunately, the number of gates, and therefore the time the quantum state must be preserved, is linear with the size of the individuals $\mathcal{O}(n)$.

Algorithm 1 Simple Quantum GA

Require: Fitness function F, crossover rate cr, mutation rate mr, generation size m, stopping condition
1: Generate m random solutions
2: **repeat**
3: **for** $i = 1 : m/2-1$ **do**
4: Select 2 parents
5: Run the quantum circuit to generate 2 children (see Alg. 2)
6: Evaluate the children fitness value
7: **end for**
8: Merge the children and the 2 best individuals into the next generation
9: **until** stopping condition is met
10: **return** The best individual

Algorithm 2 Quantum circuit description

Require: Parents $p_{1,2}$, crossover rate cr, mutation rate mr
1: $N =$ bit length of parent
2: $\theta_{cr} = 2 \arcsin(\sqrt{cr})$
3: Initialize quantum state, $|0\rangle^{\otimes 2N+2}$
4: Apply $R_y(2 \arcsin(\sqrt{cr}))$ gate to the crossover qubit
5: Apply $R_y(2 \arcsin(\sqrt{mr}))$ gate to the mutation qubit
6: **for** all bits of both parents **do**
7: **if** bit j of parent i == 1 **then**
8: Apply X gate to qubit j of parent i
9: **end if**
10: **end for**
11: **for** $j = 1 : N$ **do**
12: Apply Fredkin to qubit j of both parents as the targets with the crossover qubit as the control
13: Reset the crossover qubit and apply $R_y(2 \arcsin(\sqrt{cr}))$
14: **end for**
15: **for** all qubits of both parents **do**
16: Apply CNOT gate to the parent qubit with mutation qubit as the control
17: Reset the mutation qubit and apply $R_y(2 \arcsin(\sqrt{mr}))$
18: **end for**
19: **return** The measure of parent qubits in the z axis

IV. Experiments

To test the proposed algorithm we have realized some simulation experiments over commonly used benchmarks. More precisely, the problem to solve is to find the $x = [x_1, x_2, \ldots, x_i, \ldots, x_n]$ with $x_i = \{0, 1\}$ that minimizes a given function $F(x) = f$. We have used the same 5 benchmarking functions as in [17]:

$$F_1(x) = \sum_{i=1}^{n} x_i^2, \ |x_i| < 5.12, \ \min(F_1) = 0 \text{ for } x_i = 0, \quad (2)$$

$$F_2(x) = 20 + e - 20 \exp\left(-\frac{1}{5}\sqrt{\frac{\sum x_i^2}{n}}\right) - \exp\left(\frac{\sum \cos(2\pi x_i)}{n}\right), \quad |x_i| < 30, \ \min(F_2) = 0 \text{ for } x_i = 0, \quad (3)$$

$$F_3(x) = 10n + \sum_{i=1}^{n} \left(x_i^2 - 10\cos(2\pi x_i)\right), \quad |x_i| < 5.12, \ \min(F_3) = 0 \text{ for } x_i = 0, \quad (4)$$

$$F_4(x) = \sum_{i=1}^{n} x_i^2 + \left(\sum_{i=1}^{n} \frac{x_i}{2}\right)^2 + \left(\sum_{i=1}^{n} \frac{x_i}{2}\right)^4, \quad -5 \leq x_i \leq 10, \ \min(F_4) = 0 \text{ for } x_i = 0, \quad (5)$$

$$F_5(x) = \sum_{i=1}^{n} i x_i^2, \ |x_i| < 5.12, \ \min(F_5) = 0 \text{ for } x_i = 0, \quad (6)$$

where n is the dimension of the problem.

TABLE I
RESULTS OF THE BENCHMARKING

Functions	QGA			GA uniform	GA single-point
	Val 1	Val 2	Val 3	Mean	Mean
F1	0.0019	0.0013	0.0015	0.0066	0.0094
F2	3.6	6.8	1.4	2.9	3.3
F3	17	8.6	6.7	4.1	3.4
F4	0.0011	0.00064	0.0020	0.011	0.042
F5	0.0022	0.27	0.61	0.017	0.024

For the coding, we have used a Gray coding, achieving the maximum attainable accuracy [18, Sec 8.2]. The fitness is calculated using the all-vs-all simple sampling method [18, Sec 15.2.1]. To obtain the fitness of the initial population the individuals are compared with themselves. As the simulation of a quantum circuit has a limit on the number of qubits we can work with, we have used 30 bit solutions with dimension 3 problems. That is, we have coded each number with 10 bits.

For running the classical GA we used *Matlab R2021a*. In order to have a fair comparison, we used two versions of GA, one with the uniform and the other with the single-point crossover strategy. We repeated the algorithm 100 times, taking the mean value.

The QGA has been simulated using the *Matrix Product State* simulator from IBM [19]. As this is an ideal simulator, there is no restriction on the topology of the circuit. However, we are limited to the open access services. This limits the response times of the queries to process the simulations. To avoid overflowing the system, we have only run QGA thrice per benchmark function, with a maximum of $3\times5\times50\times9 = 6750$ circuit runs.

In order to compare the 3 versions of the algorithm, we have used the same input parameters in all of them (Tab. II). The results of the benchmarking are presented on the table I. If we compare the results we obtained with QGA to the mean values of those of GA, we see that the performance of the two is similar.

TABLE II
INPUT PARAMETERS

Length of individual (n)	30
Population size (m)	20
Crossover rate (cr)	0.7
Mutation rate (mr)	0.01
$maxIter$	50
$converIter$	10

If we wanted to run QGA in a real quantum computer, we have to take into account the topology of the circuit. In the proposed algorithm, ideally we only need 1 qubit to code the mutation and the crossover rate. However, this qubit has to be connected with all other qubits at the same time. This is far from doable in the current and near-term quantum computers. One solution to this problem is to use SWAP gates to move the qubit around the circuit. Although this is possible, each time a SWAP gate is used there is an increasing probability that the information on the qubit gets destroyed due to gate errors. To avoid this, we can instead duplicate the qubit in different positions on the circuit. This would minimize the information loss by using more qubits than the required. An example of this is shown in Fig. 2, where we have a topology of the circuit corresponds to a plane graph. This feature makes it suitable to be built with the current superconducting circuits technology.

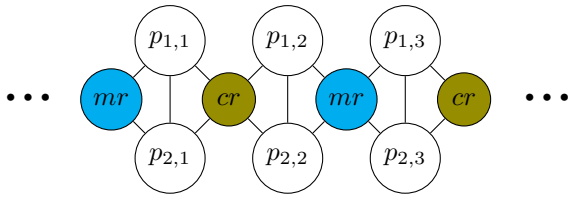

Fig. 2. Proposed circuit topology and qubit distribution for QGA. Here, each parent qubit $p_{i,j}$ is connected such way we can implement the circuit in Fig. 1 without using SWAP gates, at the cost of adding $n-2$ extra qubits.

V. CONCLUSIONS AND FUTURE PERSPECTIVES

In this paper, we present a simple algorithm that implements a quantum version of GA. The proposed QGA directly translates the crossover and mutation strategies into a quantum circuit. This algorithm does not require any oracle, leading thus to a more simple and direct implementation. Through a benchmarking of QGA against the classical GA, we are confident that both approaches are equivalent in terms of obtaining the optimal solution of a problem, up to a reasonable uncertainty.

The quantum version of GA we proposed here does not suppose any speedup in the computational complexity. However, it is shown that the algorithm can be implemented efficiently in a quantum computer. This could open the door to develop a quantum version of other heuristic or metaheuristic algorithms based on the coding of classical probabilities directly as the probability of measuring a quantum state in a certain basis. This could lead to the finding of new strategies that truly take advantage of the quantum behaviour of the system, such as superposition of states, entanglement or quantum parallelization.

REFERENCES

[1] National Academies of Sciences, Engineering, and Medicine, "Quantum Computing: Progress and Prospects," Washington, DC: The National Academies Press, 2019.
[2] C. M. Bishop, "Pattern Recognition and Machine Learning," Springer, 2006.
[3] M. A. Nielsen, I. L. Chuang, (2010), "Quantum Computation and Quantum Information (2nd ed.)," Cambridge, Cambridge University Press, 2010.

[4] D. G. Cory, N. Wiebe, C. Ferrie, C. E. Granade,"Robust Online Hamiltonian Learning", in New J. Phys., Vol. 14 (10), 2012.
[5] M. Krenn, "Automated Search for new Quantum Experiments," Physical Review Letters", Vol. 116 (9), 2016.
[6] V. Dunjko, H.J. Briegel "Machine learning & artificial intelligence in the quantum domain: a review of recent progress," Reports on Progress in Physics, vol. 81(7), 2018.
[7] L. K. Grover, "A fast quantum mechanical algorithm for database search," in Proc. 28th An. ACM STOC, pp. 212-219, 1996.
[8] M. Udrescu, L. Prodan, and M. Vlăduţiu, "Implementing quantum genetic algorithms: a solution based on Grover's algorithm," in Proc. 3th CF, pp. 71-82, 2006.
[9] A. Malossini, E. Blanzieri, and T. Calarco, "Quantum Genetic Optimization," in IEEE Trans. Evol. Comput., vol. 12, no. 2, pp. 231-241, 2008.
[10] R. Ibarrondo, "Quantum Genetic Algorithms: towards the design of evolutionary algorithms in a quantum computer," in ADDI, 2020.
[11] C. K. Li, R. Roberts, and X. Yin, "Decomposition of unitary matrices and quantum gates," in Int. J. Quant. Inf., vol. 11, no. 1, pp. 1350015, 2013.
[12] W. Shu, and B. He, "A Quantum Genetic Simulated Annealing Algorithm for Task Scheduling," in ISICA 2007: Adv. Comput. Int., vol. 4683, pp. 169-176, 2007.
[13] N. N. Hegade, K. Paul, et al, "Shortcuts to Adiabaticity in Digitalized Adiabatic Quantum Computing," in Phys. Rev. Appl., vol. 15, pp. 024038, 2021.
[14] D. E. Golberg, "Genetic Algorithms in Search, Optimization and Machine Learning," MA: Addison-Wesley, 1989.
[15] J. McHall, "Genetic algorithms for modelling and optimisation," J. Comput. Appl. Math., vol. 184, pp. 205-222, 2005.
[16] Y. C. Ho, and D. L. Pepyne, "Simple Explanation of the No-Free-Lunch Theorem and Its Implications," J. Optim. Theory Appl., vol. 1, pp. 549-570, 2002.
[17] S. P. Lim, and H. Haron, "Performance of Different Techniques Applied in Genetic Algorithm towards Benchmark Functions," In Proc. 5th ACIIDS, vol. 1, pp. 255-264, 2013.
[18] A. P. Engelbrecht, "Computational Intelligence: an introduction Second Edition," Wiley Publishing, 2007.
[19] IBM Quantum. https://quantum-computing.ibm.com/