Implementation of a Quantum Genetic Algorithm on a Variational Circuit

Simone Tomè
DEIB
Politecnico di Milano
Milano, Italy
simone.tome@mail.polimi.it

***Abstract*—Quantum genetic algorithms are an evolution of their classical counterpart that exploit quantum operations and the fuzziness of the qubit to achieve better performance in resolving optimization problems. To the best of our knowledge, the current state of the art on quantum genetics is focused on theoretical aspects but it does not consider the practical challenges of implementation on a real quantum device. This paper proposes a practical implementation that paves the way for running on a real quantum device and showcases some experimental results.**

***Index Terms*—Quantum Genetics, Genetic Quantum Algorithms, Quantum Device**

I. Introduction

Genetic Algorithms (GA) are optimization methods that rely on evolutionary theory concepts to move around the solution space of an optimization problem. They represent solutions by chromosomes (i.e. sequences of genes). Given that genes are describable by bits and that chromosomes are bit strings, a GA is applicable to an optimization problem if its solutions are representable in the form of bit strings. The basic pipeline of a genetic algorithm can be summarized by the control flow diagram shown in fig. 1.

Different existing heuristic implementations of genetic algorithms aim at selecting the best individual (solution) in an either unconstrained or constrained space of solutions. One well-known heuristics is the family of Genetic Quantum Algorithms (GQA) proposed initially by Kuk-Hyun et al. [1]. These algorithms have been applied successfully in many optimization problems, ranging from constrained project scheduling [2], combinatorial optimization [1], principal component analysis [3], and many others listed in a survey by Gexiang Zhang [4]. However, none of them provides an implementation on a real quantum device and does not even cover all of the steps in the GA canonical pipeline of fig. 1. Malossini et al. [5] proposed a refinement of GQA to resolve the latter issue. However, the physical implementation of an oracle is not trivial for an arbitrary function. Additionally, the number of gates necessary to approximate the function could be very high. It introduces noise in the circuit, making the computation unfeasible with actual NISQ (noisy intermediate-scale quantum era) technologies.

In this paper, we propose a different approach from the two versions discussed above. We exploit canonical genetic

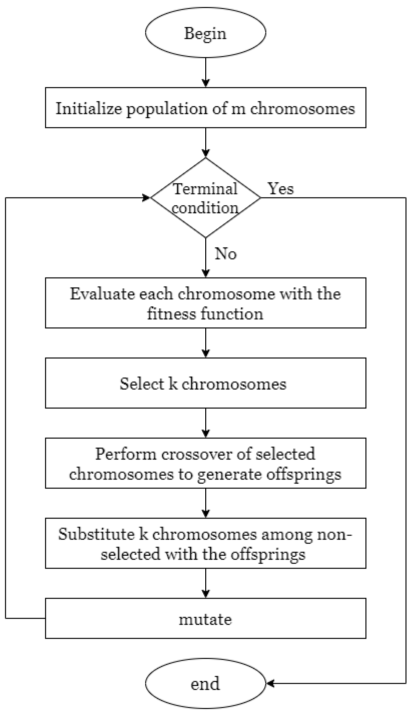

Begin
Initialize population of m chromosomes
Terminal condition
Yes
No
Evaluate each chromosome with the fitness function
Select k chromosomes
Perform crossover of selected chromosomes to generate offsprings
Substitute k chromosomes among non-selected with the offsprings
mutate
end

Fig. 1: The genetic algorithm pipeline: The *terminal condition* can either be a threshold (find a chromosome with a *fitness* greater than a certain value) or in the generations (fixed number of iterations). The *crossover* is used to mix different solutions as it mimics the reproduction in living beings: two parents generate one offspring recombining their genetic encoding. *Mutation* is used to randomly change some chromosomes (typical rate: 1%).

algorithms [6] and variational quantum circuits to (i) add crossover and mutation operations, and (ii) allow for an implementation on a quantum device. We propose a fully-fledged GQA that can run on a quantum device with sufficient amount of qubits.

However, currently accessible quantum devices do not have large number of qubits. Hence, to evaluate our approach we had to use simulation to show ts tunability while working with a large number of qubits. We first developed an ad hoc script to show our proposed algorithm's tuning; Then, we used

Qiskit [7], to simulate the behavior of real backends. Our implementation allows for tackling any problem solvable with a genetic algorithm, by tuning the parameters and the objective function to maximize.

To prove our solution is also feasible on real quantum devices, we performed experiments on real devices with a limited number of qubits; we used a variation of our algorithm (SCGQA) to make it compatible with the available number of quibits. We then compare the GQA and SCGQA implementations to highlight their performance differences.

The structure of this paper follows: section II provides basic knowledge of GQA; section III outlines our proposed GQA and argues about its feasibility on real quantum devices; section IV reports our experiments; section V discusses some observations; finally, section VI concludes.

II. Background

Genetic Quantum Algorithm

A genetic quantum algorithm (GQA) leverages the uncertainty of the qubits but does not count for the limitations of the quantum world. In GQA, a set of m chromosomes, each built with a set of n genes (i.e., qubits), forms a quantum population. The i-th chromosome can be represented as in eq. (1), where $|\phi\rangle_j^i = (\alpha_j, \beta_j)^T$ is the j-th gene of the i-th chromosome. A chromosome can be thought of as a quantum state which represents the probability distribution of collapsing into one of the 2^n possible encodings. Algorithm 1 shows the steps of GQA.

$$|\phi\rangle^i = \begin{pmatrix} \alpha_1 & \alpha_2 & ... & \alpha_n \\ \beta_1 & \beta_2 & ... & \beta_n \end{pmatrix} \quad (1)$$

Algorithm 1 The steps of the original QGA [8].

```
$t \leftarrow 0$
Initialize a quantum population $Q(t)$
Make $P(t)$, measure of every individual $Q(t) \rightarrow P(t)$
evaluate $P(t)$
select best chromosomes among $Q(t)$
**while** ($\neg$ termination condition) **do**
    $t \leftarrow t + 1$
    Update $Q(t)$ applying Q-gates: $Q(t+1) = U(t) \cdot Q(t)$
    Make $P(t)$, measure of every individual $Q(t) \rightarrow P(t)$
    Evaluate $P(t)$
    select best chromosomes among $Q(t)$
**end while**
```

A population is initialized by having all the chromosomes into a uniform superposition state and applying an Hadamard's gate to each single gene of the m chromosomes. The i-th chromosome is represented as in eq. (2).

$$|\phi\rangle^i = H^{\otimes n} |0_n\rangle = \bigotimes_{j=1}^{n} H \cdot |0\rangle_j^i = \frac{1}{\sqrt{2^n}} \sum_{k=0}^{2^n-1} |k\rangle \quad (2)$$

$P(t)$ is the set of m measured chromosomes at generation t, where each one is just a binary string of n bits. After the initialization, there will be a measurement and all chromosomes of $Q(0)$ will be mapped to $P(0)$. The measurement of a single gene $|\phi\rangle_j^i$ is simulated using a randomly generated value $x \in [0, 1]$. If $x <= \alpha^2$ then the measure will be 0, otherwise 1. Each gene is updated at the beginning of each generation (except of the first one). The proposed update uses the $R_y(\theta)$ gate with the rules stated in table I.

x_i	b_i	$f(x) \geq f(b)$?	$\alpha_i\beta_i > 0$	$\alpha_i\beta_i < 0$	$\alpha_i = 0$	$\beta_i = 0$
0	0	F	0	0	0	0
0	0	T	0	0	0	0
0	1	F	0	0	0	0
0	1	T	$-\delta\theta$	$+\delta\theta$	$\pm\delta\theta$	0
1	0	F	$-\delta\theta$	$+\delta\theta$	$\pm\delta\theta$	0
1	0	T	$+\delta\theta$	$-\delta\theta$	0	$\pm\delta\theta$
1	1	F	$+\delta\theta$	$-\delta\theta$	0	$\pm\delta\theta$
1	1	T	$+\delta\theta$	$-\delta\theta$	0	$\pm\delta\theta$

TABLE I: The **Update** procedure rules for algorithm 1. x_i is the measured gene in consideration; b_i is the measured gene of the best found chromosome; $f(\cdot)$ is the fitness function.

The original GQA [1] faces three problems if implemented on a physical quantum device [8].

- The algorithm makes intermediate measurements. Hence, its implementation would collapse the coherent state of the qubits, loosing the information carried with it.
- It is impossible to find an unitary operator to perform crossover due to the *no-cloning theorem* [9] which implies that qubits can be swapped but not copied. However, the copy is necessary for the crossover procedure, as each gene is represented by a qubit and the offsprings have the same genes as their parents.
- The update procedure of the original GQA is basically a query-table, as the one shown in table I . However, in quantum computing, we cannot observe the internal state of a qubit without collapsing its state to a basis one.

Section III explains how we bypass these problems by exploiting the potential of variational circuits [10] and by performing the crossover and the generational updates via classical computation.

III. An improved version of GQA

Some actions are very hard or even impossible on a quantum device, while they can be performed easily on a classical computer. For example, assigning a value to a register is quickly done on a classical computer just by performing a single operation. However, a unitary matrix must present a sequence of operations to do the same on a quantum device.

To benefit from both classical and quantum computation, our proposed implementation has two routines: one performed on a quantum circuit and one on a classical computer. This is possible with a variational circuit, where a quantum circuit with one or more free parameters is controlled by an external device, e.g., a classical computer. With variational circuits, we can control the rotational gates used in the original GQA. Figure 2 shows the schema that resumes the required circuit by our proposed version of GQA.

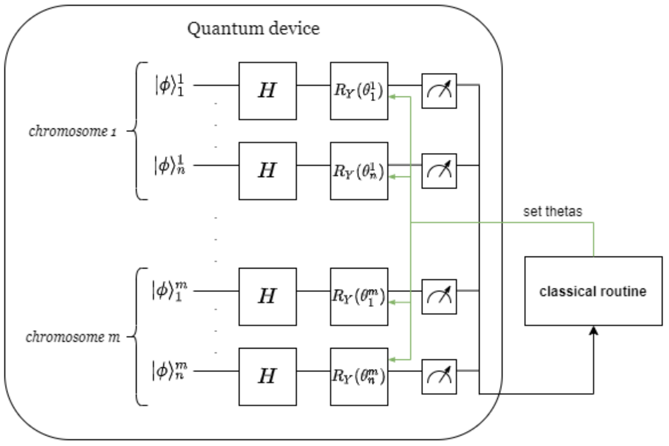


Fig. 2: Circuit schema for algorithm 2

Variable/Procedure	Comment
t	Generation
Q(t)	Quantum population at generation t
M(t)	Measured population at generation t
E(t)	Evaluations of M(t)
S(t)	Selected chromosomes
Θ(t)	Rotations of the mxn R_y gates
b	Best chromosome found
$f(\cdot)$	Black box objective function to minimize
η	Mutation rate
c	Crossover rate
shuffle(M(t))	shuffle randomly the measured chromosomes
uniform(range)	returns a random number within the range

TABLE II: The notation guide to our proposed GQA. Note that $f(\cdot)$ is equal to the inverse of the fitness function.

Algorithm 2 Implementation of GQA with additional *crossover* and *mutation* operation. An higher level perspective is given by fig. 2.

```
$t \leftarrow 0$
**while** $t < T$ **do**
                                         ▷ Quantum routine (1)
    **for** each $|\phi\rangle_i^j \in$ Q(t) **do**
        $|\phi(t)\rangle_i^j \leftarrow R_y(\Theta[i][j]) \cdot H \cdot |0\rangle$
    **end for**
    $M(t) \leftarrow measure(Q(t))$
                                         ▷ Classical routine (2)
    $(b, E(t)) \leftarrow evaluate(M(t), f(\cdot))$
    $S(t) \leftarrow select(M(t), E(t))$
    $\Theta(t+1) \leftarrow update(b, \Theta(t))$
    $x \leftarrow uniform([0,1])$
    **if** $x \leq c$ **then**
        $\Theta(t+1) \leftarrow crossover(\Theta(t+1))$
    **end if**
    $\Theta(t+1) \leftarrow mutate(\Theta(t), \eta)$
    $t \leftarrow t+1$
**end while**
```

The rest of this section explains the main steps of our proposed algorithm as depicted in algorithm 2 and table II defines the variables used in the implementation.

The Quantum Routine

The basic idea is that the superposition state of each gene could be saved if we store each increment $\pm\delta\theta$ on a classical device. To do so, we can use a data structure $\Theta \in \mathbb{R}^{m\times n}$ initialized with $m \times n$ zeros. At the first generation, random chromosomes will be measured by eq. (2) and each one will be evaluated by the classical device. For further generations the Θ data structure will contain the updated values that will modify the state of the chromosomes. We are not exploiting an oracle, often used in the quantum literature to perform optimization problems in the circuit itself, because we are dealing with unknown objective functions and we want to keep the circuit fully known. The quantum circuit treats the classical device as a black box that receives the measurements as input and produces an update of the gates' parameters.

The Classical Routine

a) Evaluation: The evaluation phase takes as input all the m measured chromosomes and evaluates them with the chosen objective function $f(\cdot)$, which depends on the problem we are trying to optimize. For example, the fitness value for the minimization of a real value function could be $z = -f(x_1, ..., x_j)$ where the variables $x_1, ..., x_j$ are represented by the chromosomes (more details in section IV).

b) Selection: A selection method could be either defined as a simple method that chooses the best chromosomes or a more sophisticated method such as roulette wheel selection, or tournament selection etc.

We used the selection method in **??** III-0b which selects $\frac{m}{2}$ chromosomes by performing small tournaments between two chromosomes, where the probability of winning is proportional to the fitness value. To avoid a naive approach, where only the best survives, this method gives higher chance of being in the next generation to the best chromosomes, and yet leaves a possibility to weaker chromosomes as well. The assigned probabilities are proportional to the fitness value of the chromosome: the chance to select one individual with fitness f_1 as to another individual with fitness f_2 is equal to $\frac{f_1}{f_1+f_2}$.

Algorithm 3 The selection method used in our experiments.

```
$C(t) \leftarrow shuffle(M(t))$
$i \leftarrow 1$
**while** $i \leq |M(t)|$ **do**
    $c_1 \leftarrow C[i]$, $c_2 \leftarrow C[i+1]$, $f_1 \leftarrow f(c_1)$, $f_2 \leftarrow f(c_2)$,
    $x \leftarrow uniform([0,1])$
    **if** $x \leq \frac{f_1}{f_1+f_2}$ **then** $S \leftarrow S \cup (c_1)$
    **else** $S \leftarrow S \cup (c_2)$
    **end if**
    $i \leftarrow i+2$
**end while**
```

c) Update: The update of the rotational values must be done according to the best solution found in the evaluation step of the algorithm. Each update must be done for a single gene, following the rules in table III, similar to the ones in the original GQA with the difference that we cannot access directly to the sign of the amplitudes α and β of each gene (qubit).

x^i_j	b_j	Update
0	0	0
0	1	$+\delta\theta$
1	0	$-\delta\theta$
1	1	0

TABLE III: The update rules of the classical routine. x^i is the i-th chromosome; b is the best chromosome of the considered generation (with the best $f(\cdot)$ value); Index j indicates the gene; $\Theta \in \mathbb{R}^{m\times n}$; $\delta\theta$ is a hyperparameter chosen a priori that affects the convergence of the algorithm.

The effects of the different hyperparameters will be seen in the experiments section IV.

The rules of table III update each value $\theta^i_j = \Theta[i][j]$ with an incremental approach. An example is the following: if the check for x^i_j returns an update of $+\delta\theta$ then $\theta^i_j(t+1) = \theta^i_j(t) + \delta\theta$.

This workaround allows to store the amplitudes α^i_j and β^i_j as they're just the result of $|\phi(t)\rangle^i_j = R_y(\theta(t)^i_j)H|0\rangle$ and being θ^i_j the only free parameter in the circuit, chosen by the classical device, we can know α^i_j and β^i_j as the parameters and their initialization is fully known ($\theta(0)^i_j = 0, \forall i, j \in \{(1..n) \times (1..m)\}$).

d) Crossover: The proposed method mixes the rotations of Θ before changing the rotations of the R_y gates directly on the classical device, as opposed to mixing the genes on the circuit (using swap gates controlled by ancillae). For example, consider two chromosomes that have been selected to crossover and suppose having $n = 4$ genes. Each of this two chromosomes have their own rotations (for the R_y gates) stored during previous steps in Θ. *Chromosome 1* has : $\theta^1_1, \theta^1_2, \theta^1_3, \theta^1_4$ while *Chromosome 2* has : $\theta^2_1, \theta^2_2, \theta^2_3, \theta^2_4$. A possible child chromosome could be $\theta^1_1, \theta^1_2, \theta^2_3, \theta^2_4$. Reminding that genes are in the form of $|\phi(t)\rangle^i_j = R_y(\theta(t)^i_j)H|0\rangle$ at the end of the quantum circuit, then:

$$\textit{Chromosome 1}\text{: } \begin{pmatrix}|\phi\rangle^1_1 & |\phi\rangle^1_2 & |\phi\rangle^1_3 & |\phi\rangle^1_4\end{pmatrix}$$
$$\textit{Chromosome 2}\text{: } \begin{pmatrix}|\phi\rangle^2_1 & |\phi\rangle^2_2 & |\phi\rangle^2_3 & |\phi\rangle^2_4\end{pmatrix}$$
$$\textit{Chromosome child}\text{: } \begin{pmatrix}|\phi\rangle^1_1 & |\phi\rangle^1_2 & |\phi\rangle^2_3 & |\phi\rangle^2_4\end{pmatrix}$$

The crossover operation's frequency is chosen a priori by tuning the crossover rate c.

e) Mutation: It is performed by negating a gene (i.e., 0 becomes 1, 1 becomes 0). Mutating a quantum gene is done using an X-gate to invert the amplitudes α and β [8]. This sounds trivial in theory, however it is not trivial in the physical sense as we would need $m \times n$ qubits to control $m \times n$ X-gates to perform a mutation according to a mutation rate. More gates and qubits would increase the noise in the circuit, so the easiest way is to perform the mutation in the classical environment by inverting the rotations. If the gene j of the i-th chromosome mutates, then $\theta^i_j(t+1) \leftarrow -\theta^i_j(t)$. It has the same effect of adding an X-gate before the measurement of the mutated gene, but making it on the classical side does not affect the circuit complexity.

IV. Experiments

This section reports our experiments in two parts: results achieved by simulation using the Qiskit library [7], and results from experiments on *IBM* real quantum device [11]. We decided to use both evaluation approaches because the publicly accessible quantum devices have an insufficient amount of qubits for "sufficiently difficult" problems which would have limited the extent of our evaluation if we only relied on experimenting with real devices.

More specifically, the crossover operation imposes the number of chromosomes to be at least four, i.e., two parents that generate two offspring. This means that to solve a problem whose solution must be encoded in an 8-bit string, we need at least 32 qubits. However, there is no real quantum device accessible with this number of qubits and we have to use simulators to test the algorithm on larger instances. Thus, we decided to use mock backends that generate simulations using noise values taken from real IBM quantum devices.

Additionally, to proof of the feasibility of our proposal on current NISQ architectures, we present a modified version of the implementation for the single chromosome variant (SCQGA). This variation is executable with five qubits on an openly accessible real quantum device.

A. Simulation Results

The Tuning of the Proposed GQA: The problem of function optimization can be tackled with genetic algorithms. A chromosome can easily represent a real value and the level of precision can be tuned by increasing or decreasing the number of genes. Let's take for example the problem of the optimization of a function $f(x, y)$ with a chromosome of length n (bit-string of length n). If the variables are bounded (by u and l), the real value of x (obviously the same holds for other variables) could be expressed with eq. (3), where x_i is the i-th gene.

$$\begin{aligned} x &= u_x\alpha + l_x(1-\alpha) \\ \alpha &\in [0,1] \ \wedge x \in [l_x, u_x] \\ \alpha &= \sum_{i=1}^{n} 2^{-i} \times x_i \end{aligned} \tag{3}$$

This, in turn, would allow for expressing the real values with a bit-string chromosome. If we have more than one variable, we just have to split the chromosome (i.e. for a forty genes chromosome, the first twenty genes encode the first variable, while the second twenty encode the second variable). The more the genes, the more precise is the encoding. To test our algorithm, we took into consideration non-convex functions with different numbers of optimal points.

We used the functions listed in table IV for our experiments which are a suitable test bench for the algorithm because of their multiple optimal and/or sub-optimal points.

Function	$f(x,y)$	**x Interval**	**y Interval**
Peaks [12]	$3(1-x)^2e^{(-(x^2)-(y+1)^2)} - 10(\frac{x}{5} - x^3 - y^5)e^{(-x^2-y^2)} - \frac{1}{3}e^{(-(x+1)^2-y^2)}$	[-3,3]	[-3,3]
Eggholder [13]	-$(y+47)sin(\sqrt{\lvert y+\frac{x}{2}+47\rvert}) - xsin(\sqrt{\lvert x-y-47\rvert})$	[-512,512]	[-512,512]
Rastrigin [14]	$20 + x^2 + y^2 - 10(cos(2x\pi) + cos(2y\pi))$	[-5.12,5.12]	[-5.12,5.12]

TABLE IV: Functions used for experiments.

We ran the simulations by a python script available at our repository [15], using the parameters in table V. Parameter *m* is the number of chromosomes and *n* represents the number of genes for each chromosome. The problem tries to optimize real values (x, y) and this means that for each real value we have $2^{\frac{n}{2}}$ discrete values in between the defined bounds of eq. (3). The parameter values are selected with trial and error starting from the proposed values in the Kuk-Hyun et al. [1] work. The only value that is far from the original parameter is for the number of individuals, which has been reduced to 16 to lower the computation time. The script aims to simulate the high-level behavior of the proposed circuit without regarding the noise introduced by gates or limiting the number of qubits. It doesn't represent the collective state (which will use an exponential amount of memory) by creating the complete hamiltonian. Instead, it simulates the circuit qubit-by-qubit using simple matrix multiplication for each one and collapses its state after a measurement. The advantage of this method is the ability of simulating problems where a large number of qubits (e.g., thousands) are needed, at the cost of acting in an ideal environment, where qubits are stochastic variables and not subject to noise.

Parameter	**Value**
m	16
n	64
generations	200
mutation rate	1%
crossover rate	50%

TABLE V: Fixed parameters for the tests shown in figs. 3 to 6.

function	**0.0025**π	**0.025**π	**0.25**π	**0.5**π
Peaks	-6.4460	-6.5282	-6.2888	-5.9826
Eggholder	-915.5172	-929.2570	-864.0590	-845.2702
Rastrigin	0.7267	0.1915	0.6738	1.6058

TABLE VI: Average results to synthetize the results that could be seen in fig. 3 after 200 generations.

Figure 3 shows the value of each function in table IV for the best found chromosomes, using different $\delta\theta$ values. The results are summarized in table VI and are obtained by taking the medium of 50 different runs, each one running for 200 generations. Figures 4 to 6 show the effect of different $\delta\theta$. Both *mutation* and *crossover* rate have been set to 0% to better reflect how $\delta\theta$ affects the algorithm performance. It is evident how this parameter is able to adjust the trade off between exploration and exploitation. Higher values of $\delta\theta$ tend to make the next generations exploit the current best solution, while lower values tend to make smaller changes to the genes state. This implies that the next generation will be more similar to the previous one, providing a more explorative search. In the plots, the exploration magnitude is reflected by the amount of optima points (colored in dark blue) visited through the generations. Different *X* markers are used to point the best chromosome for each generation. These plots only show the first 100 generation of *X*.

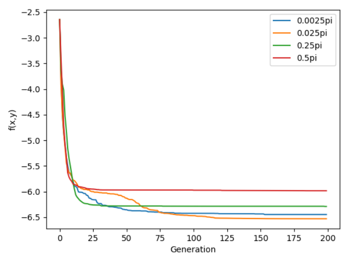
Other experiments that highlight the effect of crossover and mutation are reported in table VII, where 8 and 16 individuals have been used. The experiments show that having an adjustable parameter for both the crossover and mutation rate may improve the performance w.r.t. a version without such operators, at least for the tested functions.

The reported experiments above demonstrate that the algorithm with the incremental update approach is able to optimize an objective function in an unconstrained setting.

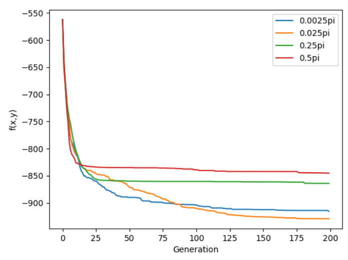
V. Discussions

Experiments showed that the implementation of the GQA works well on a real device if a sufficient amount of qubits is provided; However, the results obtained with the single chromosome variant suggest that at the moment we may trade the possibility of the crossover operation with the possibility of solving problems that requires a longer encoding and consequently more qubits. For example, assume a problem that requires to encode the solution in a 64-bit string; The original proposed GQA needs at least 256 qubits, while with the single chromosome variant uses only 64 qubits.
We claim that our proposed GQA runs the same as m parallel single chromosome variants; Yet, the update procedure is done globally using the best overall solution and, unlike the single chromosome variant, it doesn't need to estimate the currently expected chromosome.

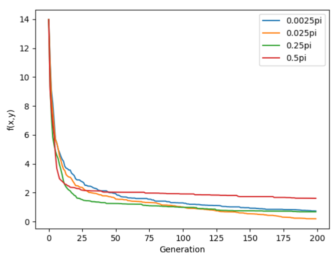
One might ask why not just sticking to the classical computation to simulate everything? The implementation on a real device speedup the process of updating each rotational value, because it is done in parallel. Consider a one core model and a single chromosome variant. The update of the rotational value for each qubit (at the beginning of each generation) has a cost of $\Theta(n)$. However, if it is done in parallel by the circuit, that becomes $\Theta(1)$. This doesn't change the asymptotic behavior because to evaluate the fitness of each measurement, we still have to use the classical device. The circuit itself is very



(a) Peaks

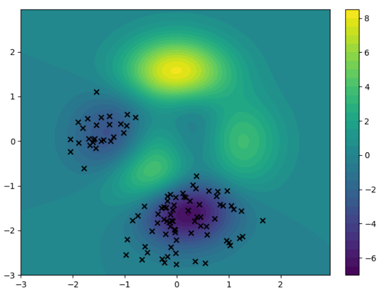


(b) Eggholder

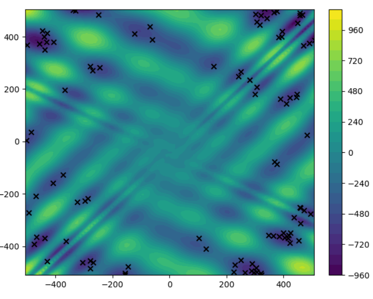


(c) Rastrigin

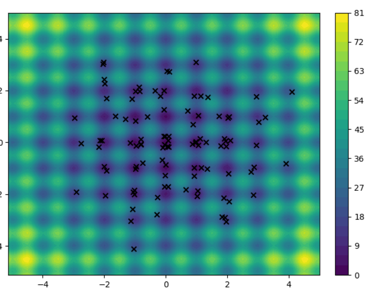
Fig. 3: Function value over generations averaging 50 runs for each parameter's value.



(a) Peaks

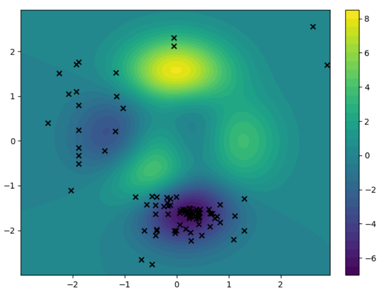


(b) Eggholder

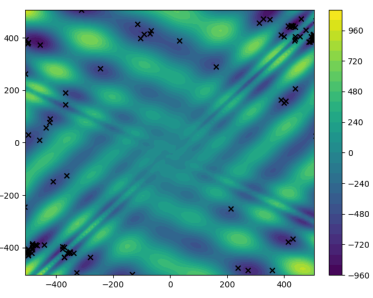


(c) Rastrigin

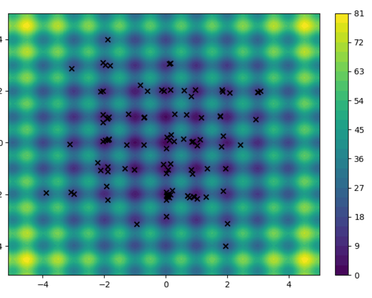
Fig. 4: Best chromosomes with $\delta\theta = 0.0025\pi$



(a) Peaks

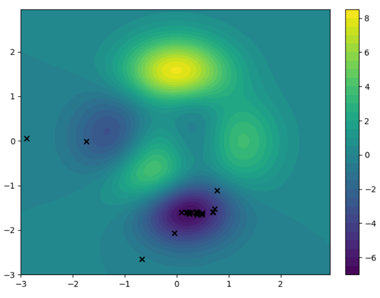


(b) Eggholder

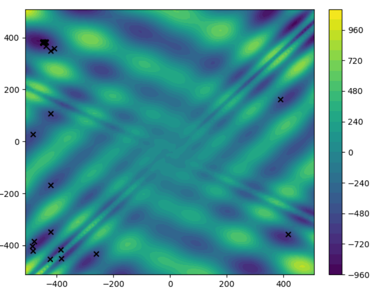


(c) Rastrigin

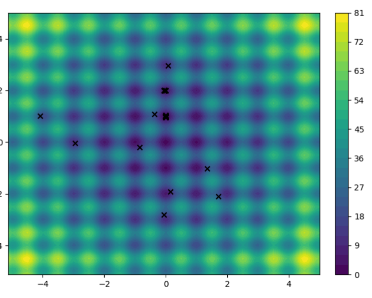
Fig. 5: Best chromosomes with $\delta\theta = 0.025\pi$



(a) Peaks



(b) Eggholder



(c) Rastrigin

Fig. 6: Best chromosomes with $\delta\theta = 0.25\pi$

	8 individuals											
	Peaks				**Eggholder**				**Rastrigin**			
	0%	25%	50%	75%	0%	25%	50%	75%	0%	25%	50%	75%
0%	-6.4858	-6.4492	-6.4329	-6.3400	-910.8187	-915.7416	-906.4790	-908.4435	0.7322	0.8714	0.9675	0.8446
1%	-6.4872	-6.4994	-6.5001	-6.4826	**-930.6188**	-930.0463	-914.6807	-914.5152	0.4695	0.5593	0.4809	0.6681
2.5%	**-6.5442**	-6.5341	-6.5270	-6.5349	-923.8602	-926.2239	-926.91786	-929.3611	0.8660	0.6943	**0.4605**	0.5837
	16 individuals											
	Peaks				**Eggholder**				**Rastrigin**			
	0%	25%	50%	75%	0%	25%	50%	75%	0%	25%	50%	75%
0%	-6.5116	-6.3931	-6.3934	-6.4458	-927.3270	-923.4870	-937.0828	-922.4733	0.3518	0.3896	0.4466	0.6468
1%	-6.5427	-6.5047	-6.5299	-6.5352	-931.4131	-931.5955	-927.0126	-928.1251	**0.0667**	0.1373	0.2167	0.5009
2.5%	-6.5434	**-6.5492**	-6.5333	-6.5430	-936.7344	**-939.0036**	-933.92074	-937.4145	0.3100	0.2152	0.1805	0.1427

TABLE VII: Performances averaged over 30 runs for 200 generations with 8 and 16 individuals and $\delta\theta = 0.025\pi$. The parameters on the row and column are *mutation* and *crossover* rates respectively. The bold entries are the best results per function.

general and adapts to any kind of optimization problem. So, adding components will be easy but, in order to remain as general as possible, we preferred to delegate specific tasks to the classical device, where it's easier to formulate the objective function and the logic of the implementation.

VI. Conclusions

This paper provides an implementation for both the GQA and the single chromosome variant (SCGQA) compliant with the quantum computing principles. We then explored the behavior of the two implementations: the first has been tested with a simulator (both an ideal and a noisy one) due to the high qubits requirement, while the single chromosome variant has been tested on a real device. We discussed in section V how the latter implementation may be a good solution to overcome the requirement of having a number of qubits at least 4 times the length of the solution's encoding. We also provided results over two different problems: a bounded function optimization and a combinatorial optimization with an ad-hoc repair routine to explore the constrained solution space. Further works could make improvements on the implementation to solve more constrained optimization problems (the *repair* algorithm - **??** - it's for Knapsack) and test them on quantum devices with a larger number of qubits. This would be interesting to see if the noise introduced by a bigger circuit disrupts or not the performances of the two implementations on bigger instances.

References

[1] K.-H. Han and J.-H. Kim, "Genetic quantum algorithm and its application to combinatorial optimization problem," in *Proceedings of the 2000 Congress on Evolutionary Computation*, vol. 2, 2000, pp. 1354–1360 vol.2.

[2] H. M. H. Saad, R. K. Chakrabortty, S. Elsayed, and M. J. Ryan, "Quantum-inspired genetic algorithm for resource-constrained project-scheduling," *IEEE Access*, vol. 9, pp. 38 488–38 502, 2021.

[3] Y. Shen, W. Nai, Y. Li, J. Shen, Z. Yang, D. Li, and Y. Xing, "Principal component analysis based on quantum genetic algorithm with t-distribution parameters," in *2021 IEEE 5th Advanced Information Technology, Electronic and Automation Control Conf. (IAEAC)*, vol. 5, 2021, pp. 2378–2382.

[4] G. Zhang, "Quantum-inspired evolutionary algorithms: a survey and empirical study," *Journal of Heuristics*, vol. 17, no. 3, pp. 303–351, Jun 2011.

[5] A. Malossini, E. Blanzieri, and T. Calarco, "Quantum genetic optimization," *IEEE Transactions on Evolutionary Computation*, vol. 12, no. 2, pp. 231–241, 2008.

[6] D. Whitley, "A genetic algorithm tutorial," *Statistics and Computing*, vol. 4, pp. 65–85, 1994.

[7] "Qiskit: An open-source framework for quantum computing," zenodo/qiskit, 2021.

[8] R. Lahoz-Beltra, "Quantum genetic algorithms for computer scientists," *Computers*, vol. 5, no. 4, 2016.

[9] W. K. Wootters and W. H. Zurek, "A single quantum cannot be cloned," *Nature*, vol. 299, no. 5886, pp. 802–803, Oct 1982.

[10] J. R. McClean, J. Romero, R. Babbush, and A. Aspuru-Guzik, "The theory of variational hybrid quantum-classical algorithms," *New Journal of Physics*, vol. 18, no. 2, p. 023023, feb 2016.

[11] "IBM Quantum," ibm/quantum, 2021.

[12] MathWorks, "MATLAB Peaks function," matlab/peaks, last visited 2021/10/28.

[13] C. Vanaret, "Hybridization of interval methods and evolutionary algorithms for solving difficult optimization problems," Ph.D. dissertation, Institut National Polytechnique de Toulouse (France), 01 2015.

[14] M. A. Potter and K. A. De Jong, "A cooperative coevolutionary approach to function optimization," in *Parallel Problem Solving from Nature — PPSN III*, Y. Davidor, H.-P. Schwefel, and R. Männer, Eds. Berlin, Heidelberg: Springer, 1994, pp. 249–257.

[15] S. Tomè, "The script of quantum genetic algorithm on a variational circuit," github/QGA, 2021.