Gradu Amaierako Lana / Trabajo Fin de Grado
Fisikako Gradua / Grado en Física

Quantum Genetic Algorithms

Towards the design of evolutionary algorithms in a quantum computer

Egilea/Autor/a:
Rubén Ibarrondo López
Zuzendaria/Director/a:
Prof. Iñigo L. Egusquiza
Zuzendarikidea/Codirector/a:
Dr. Mikel Sanz

Leioa, 2020ko ekainaren 17a /Leioa, 17 de junio de 2017

Contents

Introduction and objectives

Quantum computation is a computational paradigm which makes use of quantum phenomena, such as quantum superposition and interference, in order to obtain algorithms with speed-ups unattainable in a classical computer. Although implementations in physical platforms are relentlessly progressing, achieving a quantum processor sufficiently large to tackle realistic problems is undoubtedly challenging and will take some years, maybe a decade. Additionally, not only is the implementation challenging, but also the development of algorithms which are often quite counterintuitive.

Bioinspired algorithms are computational tools that mimic certain natural processes. For instance, genetic algorithms are inspired in the principles of Darwinian evolution in order to find robust solutions to a constrained optimisation problem. They are typically used when the implementation of the constraints is cumbersome within the framework of other optimisation methods and a robust approach is required. The naturally subsequent question is whether it is possible to enhance quantum mechanically these bioinspired algorithms.

The development of a quantum genetic algorithm is undoubtedly intriguing . On the one hand, we face the challenge of creating a crossover [1] quantum subroutine which does not violate the no-cloning theorem. On the other hand, either faster convergence or more robust solutions might be obtained. Additionally, genetic algorithms are straightforwardly distributable. Therefore, they are natural candidates as algorithms in distributed quantum computation, which may be implemented in available noisy intermediate-scale quantum (NISQ) processors.

In this work, we pursue three main goals. The first one is to understand and explain some fundamental concepts of quantum computation, such as the qubit, quantum circuits and quantum states, among others. The second one is to explain the principles of genetic algorithms and analyse the effect of the different choices available to us, such as the codification or the choice of parameters. The third objective merges the previous concepts, aiming at stablishing the first steps towards the design of quantum genetic algorithms. The work required developing Python 3 programs which are available in the repository of the University of the Basque Country (ADDI).

Along these lines, this work is divided into three chapters, apart from the introduction and conclusions. In the first chapter, we review the grounds of quantum computation, as well as some relevant quantum algorithms. The second one revisits the fundamentals of genetic algorithms and the influence of the codification, the selection method and the choice of parameters in the performance of these algorithms. Finally, in the last chapter, we explain two different approaches for the design of a quantum genetic algorithm. In the first one, we revisit a previous work which employs a quantum algorithm in one of the subroutines of a classical genetic algorithm in order to speed it up. We also explore

a possible variation of this algorithm which shows advantage in certain regimes. Finally, we develop the first building blocks of a fully quantum genetic algorithm. This paves the way for he future design of distributable quantum algorithms based on genetic algorithms, which might show quantum advantage in respect to classical ones.

Chapter 1

Fundamentals of Quantum Computation

Quantum computation is a novel computational paradigm which employs quantum entanglement and superposition as resources to accelerate classical algorithms. This chapter reviews the fundamentals of quantum computation and quantum information required to understand the rest of the work and it it is mainly based on Ref. [2].

This introduction is mainly addressed to readers that are not familiar with quantum computation. Firstly I review the concepts of qubit, quantum gate, quantum measurement and quantum circuit notation (Section 1.1). Finally, I revisit in the next sections the quantum no-cloning theorem (Section 1.2) and quantum searching algorithms (Section 1.3), which will be of great importance to understand the final chapter.

1.1 Fundamentals of quantum computation

1.1.1 The qubit

The bit is a two-state classical system that represents the fundamental unit of classical computational information, these states are denoted by 0 or 1. The qubit is its quantum counterpart, it is a two-state quantum system, which states are denoted by $|0\rangle$ or $|1\rangle$, following Dirac notation. There is a clear difference between them: superposition. While the classical bit can only be in one of those states, for the quantum bit the most general state reads as

$$|\psi\rangle = \cos\frac{\theta}{2}|0\rangle + e^{i\varphi}\sin\frac{\theta}{2}|1\rangle\,, \tag{1.1}$$

with θ and ϕ any real numbers in the range $[0, 2\pi)$. The states $|0\rangle$ and $|1\rangle$ are known as *computational basis states.*

While the qubit is allowed to evolve following quantum mechanics, it may keep the superposition, but if it is measured it may collapse. The probability to measure each state is the square modulus of its amplitude, here $p_0 = \cos^2\frac{\theta}{2}$ and $p_1 = \sin^2\frac{\theta}{2}$. Once a quantum state is measured, it collapses to the measured state, leaving $|0\rangle$ if 0 was obtained or $|1\rangle$ if 1 was obtained.

When two bits are combined we get four possible values: 00, 01, 10 and 11. We also get four states in the computational basis for the qubit. And the most general two-qubit state can be described by complex numbers a, b, c and d, satisfying $|a|^2 + |b|^2 + |c|^2 + |d|^2 = 1$,

$$|\psi\rangle = a\,|00\rangle + b\,|01\rangle + c\,|10\rangle + d\,|11\rangle\,. \tag{1.2}$$

When many qubits are involved, another quantum property arises: entanglement. If the global state of the two-qubits can be described by the tensor product of two single qubit states, it is called separable. Otherwise, the state is called entangled. The computational-basis states satisfy it, that is $|01\rangle = |0\rangle \otimes |1\rangle$. But also $|0\rangle \otimes (|0\rangle + |1\rangle)/\sqrt{2} = (|00\rangle + |01\rangle)/\sqrt{2}$ or $(|0\rangle - |1\rangle)/\sqrt{2} \otimes |0\rangle = (|00\rangle - |10\rangle)/\sqrt{2}$ are examples of this, where we can safely define the state of a qubit without considering the other.

This cannot be done in states like $(|00\rangle + |11\rangle)/\sqrt{2}$ or $(|01\rangle - |10\rangle + |11\rangle)/\sqrt{3}$. In entangled states, measuring one of the qubits alters the information we have about the others.

1.1.2 Quantum logic gates

Classical logic gates are the representation of Boolean functions. They describe operations performed in bits that yield a 1 or a 0 as a classical result. Take the AND gate as an example, it gets a two bit input and returns a 1 if and only if both are 1, otherwise it returns a 0. This is described in its truth table shown in Fig. 1.1 where other classical gates are also described with their circuit representation.

AND

in	out
0 0	0
0 1	0
1 0	0
1 1	1

EXOR

in	out
0 0	0
0 1	1
1 0	1
1 1	0

NOT

in	out
0	1
1	0

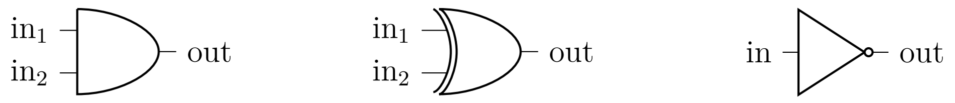


Figure 1.1: Truth tables and circuit representations for some classical gates.

These logic gates are straightforwardly implemented with electronic circuits. The 0 or 1 binary states can be represented as two-level tensions or currents, and gates can be implemented with transistors. As a natural extension, a practical manner to manipulate quantum states of qubits is needed.

According to quantum mechanics a quantum state evolves following Schrödinger equation,

$$|\psi(t)\rangle = U\,|\psi(0)\rangle\,, \qquad U \equiv \exp\left[-i\frac{Ht}{\hbar}\right], \tag{1.3}$$

where H is the time independent Hamiltonian of the system, a Hermitian operator, and thus the operator U is a unitary. With a proper Hamiltonian and time lapse, we can implement any U. In order to describe operations in a n-dimensional Hilbert space, we must operate with $2^n \times 2^n$ matrices.

The unitary operators that describe the evolution of the state are known as quantum logic gates or quantum gates. Notice that unitarity implies that the quantum logic gates are reversible. It is worth mentioning that non-reversible computation can be represented with reversible gates, which supports the universality of quantum computation [2].

Single qubit gates

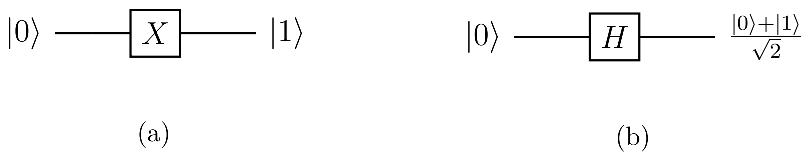


Figure 1.2: Circuit notation examples for a) X gate, also known as NOT gate and b) Hadamard gate, H.

Quantum gates acting on a single-qubit are known as single qubit gates. For a single qubit, the operation performed by the quantum evolution can be described by a 2×2 matrix. The classical NOT gate, described in Fig. 1.1, can be implemented with the Pauli-X gate

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad \text{where} \quad |0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \text{ and } |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}. \tag{1.4}$$

The action of these operators in qubits is usually represented in circuit notation. In Fig. 1.2a an example for the X gate is shown. Quantum gates enable operations which are not possible with classical bits. As an example, the Hadamard gate, H, can create a superposition of the $|0\rangle$ and $|1\rangle$ states, a Pauli-Z gate can introduce a -1 phase in the $|1\rangle$ state. Some usual single qubit gates are

$$H = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}, \quad Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}, \quad Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \quad \text{and} \quad S = \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix}. \tag{1.5}$$

When single qubit gates act on several qubits their matrix representation can be computed with the Kronecker product of matrices. For instance, if the identity gate is applied in the first qubit and a Hadamard gate is applied in the second one this can be described by

$$\mathbb{I} \otimes H = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & -1 \end{pmatrix}. \tag{1.6}$$

Two qubit gates

Although single qubit gates can act in different qubits, they operate separately. Multi-qubit gates, in turn, act on many qubits simultaneously. Let us take the case of a two-qubit circuit aimed to change the value of the second qubit, target qubit, depending on the first one, control qubit. If the control qubit is in $|q_1\rangle = |0\rangle$ state, the other qubit will not be touched, while if it is in $|q_1\rangle = |1\rangle$ the target will be flipped, apply a X

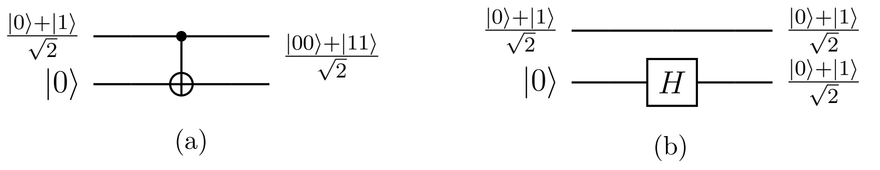
gate. This two-qubit gate is known as the CNOT gate, and it can be written in the computational basis ($\{|00\rangle, |01\rangle, |10\rangle, |11\rangle\}$ for $|q_1q_2\rangle$) as

$$U_{\text{CNOT}} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}, \tag{1.7}$$

and it is depicted in Fig. 1.3a in circuit notation. If the state of $|q_1\rangle$ was known beforehand to be 0 or 1, this gate could be implemented with the classical logic-like circuits shown in Fig. 1.4, activating or deactivating the X gate depending on the value of the control qubit.

Besides the obvious fact that we do not usually know the state of the control qubit, this approach dismisses many important cases. For instance, this classical approach would not work if the initial state in the control qubit was a superposition, such as $|q_1\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$. We could guess that such a superposition generates a similar superposition in the target qubit, thus apply an H gate instead of X, but that is not correct. In Fig. 1.3, a simple example illustrates the difference between both cases. Note that if we apply a CNOT gate on a two-qubit state which may initially be separable, as in Fig. 1.3a, we can get an entangled state.

There are also multi-qubit gates, which can act in the whole quantum system comprising an arbitrary number of qubits.



$\frac{|0\rangle+|1\rangle}{\sqrt{2}}$
$|0\rangle$
$\frac{|00\rangle+|11\rangle}{\sqrt{2}}$
(a)
$\frac{|0\rangle+|1\rangle}{\sqrt{2}}$
$|0\rangle$
H
$\frac{|0\rangle+|1\rangle}{\sqrt{2}}$
$\frac{|0\rangle+|1\rangle}{\sqrt{2}}$
(b)

Figure 1.3: Difference between a) applying a controlled-not with a superpostion state in the control and b) applying a Hadamard gate in the target qubit. Note that in both cases a superposition is obtained in the output state, but only the CNOT creates an entanglement.

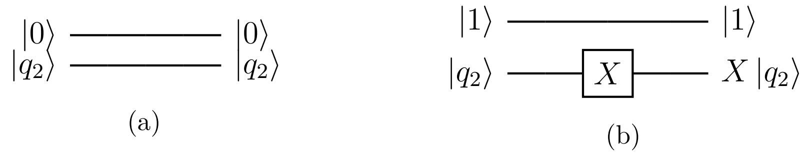


Figure 1.4: Desired behaviour for a controlled qubit flip. a) when first qubit is in $|0\rangle$ state no change in qubit two, but b) when qubit one is in $|1\rangle$ state a flip is performed, that is $|0\rangle \rightarrow |1\rangle$ or $|1\rangle \rightarrow |0\rangle$.

Uniform superposition. It is a common step in quantum algorithms to initialize the state with a uniform distribution. Nevertheless, it is usually assumed that the natural initial state for a qubit is the $|0\rangle$ state. Thus, how can it be transformed from

$|00..00\rangle$ to $(|00...00\rangle + |00...01\rangle ... + |11...11\rangle)/\sqrt{2^N}$? Or in matrix notation,

$$\begin{pmatrix}1\\0\\\vdots\\0\end{pmatrix} \xrightarrow{T} \frac{1}{\sqrt{2^N}}\begin{pmatrix}1\\1\\\vdots\\1\end{pmatrix}. \tag{1.8}$$

It is straightforward to obtain this transformation by applying the H gate in every qubit, thus $T = H_1 \otimes H_2 \otimes ... \otimes H_N$.

CNOT basis transformation. This example shows the key importance of the basis in which gates work. As an example, the CNOT gate was thought to have a control qubit and a target qubit, but that works strictly in the computational basis. If the CNOT gate is transformed to operate in a different local basis, its effect in the computational basis can vary. In order to change the basis of an operator, a unitary transformation is needed $PU_{\text{CNOT}}P^\dagger$. Considering, for instance, the case of $P = H_1 \otimes H_2$, $P^\dagger = P$, then,

$$P = H_1 \otimes H_2 = \frac{1}{2}\begin{pmatrix}1 & 1 & 1 & 1\\1 & -1 & 1 & -1\\1 & 1 & -1 & -1\\1 & -1 & -1 & 1\end{pmatrix}, \quad PU_{\text{CNOT}}P^\dagger = \begin{pmatrix}1 & 0 & 0 & 0\\0 & 0 & 0 & 1\\0 & 0 & 1 & 0\\0 & 1 & 0 & 0\end{pmatrix}, \tag{1.9}$$

which is in fact a CNOT gate, but swapping the control and target qubits. The circuit representation for this identity is shown in Fig. 1.5.

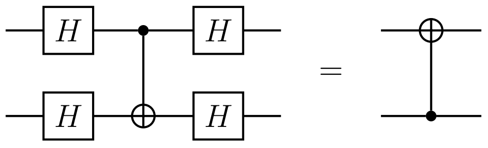


Figure 1.5: Applying a unitary transformation to the CNOT gate we can change the control and target qubits.

1.1.3 Measurements

We already pointed out that the amplitude of a state is related to its probability to be measured. Then, through measurements information about the quantum state can be extracted. Measurement implies plugging a quantum system to classical instrumentation, which implies an evolution that does not strictly follow the quantum evolution stated in Eq. 1.3. In fact, it performs a non-reversible process, causing the system to collapse according to the measured magnitudes value. When the general qubit state in Eq. 1.1 was described we mentioned that the probability of measuring each possible outcome in the computational basis depends on its amplitude, which can be used to infer the parameter θ for the state.

To obtain φ, a gate H can be applied before measuring, that is,

$$H|\psi\rangle = \frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\1 & -1\end{pmatrix}\begin{pmatrix}\cos\frac{\theta}{2}\\e^{i\varphi}\sin\frac{\theta}{2}\end{pmatrix} = \frac{1}{\sqrt{2}}\begin{pmatrix}\cos\frac{\theta}{2} + e^{i\varphi}\sin\frac{\theta}{2}\\\cos\frac{\theta}{2} - e^{i\varphi}\sin\frac{\theta}{2}\end{pmatrix}. \tag{1.10}$$

Now, the probability to measure $|0\rangle$ or $|1\rangle$ is $p_0 = (1+\cos\varphi\sin\theta)/2$ and $p_1 = (1-\cos\varphi\sin\theta)/2$, respectively. Hence, a dependence of the probabilities with φ was introduced, allowing us to infer its value. The procedure to obtain the whole information about a quantum state is known as *quantum state tomography*, and it implies a huge number of measurements in many different bases when many qubits are involved. Determining the whole state is a expensive task in terms of computational time and it can spoil any quantum advantage previously obtained in many algorithms.

Let us show an example of a measurement with two qubits in the 2-qubit state given by Eq. 1.2. If the first qubit was measured we could get 0 with probability $|a|^2 + |b|^2$ and 1 with $|c|^2 + |d|^2$, each leaving the system in different states

$$0 \to |0\rangle \otimes \frac{a|0\rangle + b|1\rangle}{\sqrt{|a|^2+|b|^2}} \quad \text{and} \quad 1 \to |1\rangle \otimes \frac{c|0\rangle + d|1\rangle}{\sqrt{|c|^2+|d|^2}}, \tag{1.11}$$

this logic also applies when measuring multiple qubits.

1.2 No-cloning theorem

In 1982, N. Herbert published a procedure that was supposed to enable faster-than-light communication [3]. This method relied on using a mechanism to clone an arbitrary quantum state. Let us explain how Herbert's proposal works before going into the no-cloning issue. Alice and Bob share an entangled state $|\Psi\rangle = (|01\rangle + |10\rangle)/\sqrt{2}$, Alice can operate in the first qubit and Bob in the second. If Alice would like to send a message to Bob she could choose between measuring in basis $\{|0\rangle, |1\rangle\}$ or $\{|+\rangle, |-\rangle\}$. As summarized in Table 1.1, the choice which Alice makes affects Bob's state, a difference that Bob can infer from the probabilities to measure each possible state. Nevertheless, to distinguish between both he would need to clone his qubit, as shown in the table.

Bob's result	Alice's measurement basis	
	$\{\|0\rangle, \|1\rangle\}$	$\{\|+\rangle, \|-\rangle\}$
Gets	$\|0\rangle$ prob. 1/2 $\|1\rangle$ prob. 1/2	$\|+\rangle$ prob. 1/2 $\|-\rangle$ prob. 1/2
Clones	$\|00\rangle$ prob. 1/2 $\|11\rangle$ prob. 1/2	$\|++\rangle$ prob. 1/2 $\|--\rangle$ prob. 1/2
Measures	$\|00\rangle$ prob. 1/2 $\|11\rangle$ prob. 1/2	$\|00\rangle, \|01\rangle,$ $\|10\rangle, \|11\rangle$ prob. 1/4

Table 1.1: Step by step process for faster than light communication, where $|\pm\rangle = (|0\rangle \pm |1\rangle)/\sqrt{2}$.

A posterior publication by W. K. Wootters and W. H. Zurek proved the impossibility of such a cloning procedure [4], currently known as the no-cloning theorem, invalidating the faster-than-light communication method proposed by Herbert. Here a similar proof is provided. If a copy of an unknown q-qubit quantum state is desired, an additional q-qubit register to codify the copy is needed. In one of them the data to be copied is stored, $|\psi\rangle$, and the other is in a predetermined known pure state to be overwritten by the copy, for instance $|0\rangle$. A hypothetical unitary cloning transformation U should perform the cloning action on any state, so $U|\psi\rangle|0\rangle = |\psi\rangle|\psi\rangle$. Therefore, we can also clone the state $|\phi\rangle$,

$$|\psi\rangle|0\rangle \xrightarrow{U} |\psi\rangle|\psi\rangle \text{ and } |\phi\rangle|0\rangle \xrightarrow{U} |\phi\rangle|\phi\rangle . \tag{1.12}$$

By using the property that under a unitary transformation the inner product is preserved, we calculate the inner product for both the initial states and the states after applying the cloning operation U, which leads to

$$\langle\psi|\phi\rangle \overbrace{\langle 0|0\rangle}^{1} = \langle\psi|\phi\rangle\langle\psi|\phi\rangle \;\Rightarrow\; \langle\psi|\phi\rangle = (\langle\psi|\phi\rangle)^2 \;\Rightarrow\; \langle\psi|\phi\rangle = 0 \text{ or } 1. \tag{1.13}$$

This proves that two arbitrary states cannot be cloned with a single unitary operator, unless they are the same or they are orthonormal.

Nevertheless, this proof does not forbid cloning two pure states in a given basis, as they are orthonormal. That is the case of the CNOT gate, which can clearly clone $|0\rangle$ and $|1\rangle$ states, but not $|+\rangle$, as shown in Fig. 1.3, where ideal cloning would perform something like Fig. 1.3b. We can check this with a most general case $|\Psi\rangle|0\rangle = a|0\rangle|0\rangle + b|1\rangle|0\rangle$, with $|a|^2 + |b|^2 = 1$, and compare what is obtained by cloning and by applying a CNOT,

$$U_{\text{CNOT}}|\Psi\rangle|0\rangle = a|0\rangle|0\rangle + b|1\rangle|1\rangle , \tag{1.14}$$
$$U_{\text{CLONE}}|\Psi\rangle|0\rangle = |\Psi\rangle|\Psi\rangle = a^2|0\rangle|0\rangle + ab|0\rangle|1\rangle + ab|1\rangle|0\rangle + b^2|1\rangle|1\rangle . \tag{1.15}$$

In both cases the probability to measure independently in each qubit a 0 is $|a|^2$ and a 1 is $|b|^2$. Indeed although applying a CNOT gate produces entanglement and actual cloning keeps the whole state separable, the probability distribution of measuring 0 or 1 in each qubit, separately, has been successfully transferred.

Exploring the limits of an imperfect cloning process has motivated multiple investigations. Some looking for cloning systems that worked with similar fidelity for any input state, called universal quantum cloning machines [5]. Others focus on the idea of cloning only part of the quantum information of a state, known in the bibliography as partial quantum cloning [6]. The latter consists of cloning the statistics of a state related to a given observable, that is, cloning an observable. In Ref. [7], the authors develop a unitary transformation that performs the cloning of a observable, extending previous results to larger dimensions. For instance, a CNOT gate can be employed to clone the statistics of an observable in a one qubit state.

In our main approach to develop quantum genetic algorithms in Section 3.2, a cloning mechanism will be crucial, as genetic evolution relies on a crossover process which requires some kind of information transfer.

1.3 Quantum searching algorithms

Optimisation algorithms can be understood as particular cases of searching algorithms, making the field relevant for the development of a quantum genetic optimisation algorithm. Searching in an unsorted list is an ubiquitous but hard problem. Classically the expected number of iterations to find a desired index among N different elements is proportional to the length of the set $\mathcal{O}(N)$. Quantum algorithms step ahead and provide a lower complexity of $\mathcal{O}\left(\sqrt{N}\right)$, as first pointed out by L. K. Grover [8].

Searching in an unsorted database is essentially identical to searching the appropriate index associated with the searched object. In other words, searching can be regarded as seeking for the binary string, x, representing the index that fits a given condition only when associated to the sought object. That condition can be described by a Boolean function satisfying $f(x) = 1$ for the searched element and $f(x) = 0$ for all others.

Although the original algorithm published by L. K. Grover studied the case of searching a single object, the description below is based on a extension to an arbitrary number of solutions [9].

1.3.1 The oracle

In this procedure, the availability of a gate which can distinguish between the states we are looking for is assumed. This quantum operator is known as an oracle. It acts on a qubit depending on the result for x encoded in the remaining qubits in the computational basis,

$$O\left|x\right\rangle\left|q\right\rangle = \left|x\right\rangle\left|q \oplus f(x)\right\rangle . \tag{1.16}$$

Usually the qubit is initialized in $|0\rangle$, so that $|1\rangle$ is obtained if the state is found, i.e.

$$O\left|x\right\rangle\left|0\right\rangle = \begin{cases} \left|x\right\rangle\left|0\right\rangle, & \text{if } f(x) = 0, \\ \left|x\right\rangle\left|1\right\rangle, & \text{if } f(x) = 1. \end{cases} \tag{1.17}$$

It can be also useful to initialize the oracle qubit to $(|0\rangle - |1\rangle)/\sqrt{2}$, in which case we obtain

$$O\left|x\right\rangle\left|0\right\rangle = \begin{cases} \left|x\right\rangle\frac{|0\rangle - |1\rangle}{\sqrt{2}}, & \text{if } f(x) = 0, \\ -\left|x\right\rangle\frac{|0\rangle - |1\rangle}{\sqrt{2}}, & \text{if } f(x) = 1. \end{cases} \tag{1.18}$$

This setup is known as phase oracle, as it shifts the phase of the state only if the searching condition is met $\left|x\right\rangle \rightarrow (-1)^{f(x)}\left|x\right\rangle$. This feature will be useful to develop the searching algorithm. It is worthy to remark that, although the oracle is able to *identify* the solution, it does not *know* the solution in advance. For problems as integer factorization or Sudoku puzzles, for which possible solutions are easily checked but difficult to find, a suitable oracle can be built without knowing the answer. In order to do so, the proper circuit representation for the oracle is needed. This representation is feasible, since any Boolean function $f(x)$ can be implemented, as explained in Section 3.2.5 of Ref. [2].

1.3.2 Quantum search with many solutions

In this section I explain the quantum algorithm to find one of t possible solutions, in a list of length N. We will consider the phase oracle in Eq. 1.18. In this operator the extra qubit can be neglected and we only consider the phase shift in the state. This allows us to focus on the evolution of the state of the qubits containing the possible solutions.

For this algorithm the state is initialized in a uniform distribution of all the possible computational states,

$$|\psi\rangle = \sum_{i\in A} k_0 |i\rangle + \sum_{i\in B} l_0 |i\rangle , \tag{1.19}$$

with $k_0 = l_0 = 1/\sqrt{N}$, the subspace A contains the solution states and B the remaining ones. The algorithm consists in applying the Grover operator, defined below, as many times as needed. This operator consists in the sequential application of the following gates, as shown in Fig. 1.6. First the phase oracle, O, is applied shifting the phase of the states $i \in A$. Then the Hadamard gate, H, is applied to each qubit. The next action requires shifting the phase of every state except the $|0\rangle$ state, described by $(2|0\rangle\langle 0| - I)$. Finally the Hadamard gate in each qubit is applied again. Summing up, the Grover operator reads

$$G = H^{\otimes n}(2|0\rangle\langle 0| - I)H^{\otimes n}O = (2|\psi\rangle\langle\psi| - I)O. \tag{1.20}$$

Each time the Grover operator is applied to the state the coefficients in subspaces A and B will change. By defining $\sin\theta = \sqrt{t/N}$ and performing the appropriate algebra, the evolution of the coefficients after j applications of G can be obtained. k_j denotes the coefficients for the desired solutions and l_j the coefficients of the undesired states,

$$k_j = \frac{1}{\sqrt{t}}\sin((2j+1)\theta), \tag{1.21}$$

$$l_j = \frac{1}{\sqrt{N-t}}\cos((2j+1)\theta). \tag{1.22}$$

Thus the desired number of applications of the Grover operator, $j \to m$, is obtained from the best possible outcome $k_m \to 1$ and $l_m \to 0$. This can be approximated by $m = \lfloor(\pi - 2\theta)/4\theta\rfloor$, rounded to the nearest smaller integer. The setup in circuit representation is depicted in Fig. 1.7.

Considering that $\theta \geq \sqrt{t/N}$ the upper bound for m is

$$m \leq \frac{\pi - 2\theta}{4\theta} < \frac{\pi}{4\theta} \leq \frac{\pi}{4}\sqrt{\frac{N}{t}}. \tag{1.23}$$

In Ref. [9] the upper bound for m is reduced to $0.5827\sqrt{N/t}$.

1.3.3 Quantum search for an unknown number of solutions

The performance of previous result depends strongly on the number of solutions t. In Fig. 1.8, the evolution of the probability of the desired states with the number of Grover steps is shown, for different t/N values. Although the initial steps always increase the desired probabilities for $t \ll N$, an excess of applications may jeopardize the chances of finding one of the desired states.

M. Boyer, G. Bassard, P. Høyer and A. Tapp developed the following algorithm which works efficiently when the number of solutions is unknown [9], but restricted to

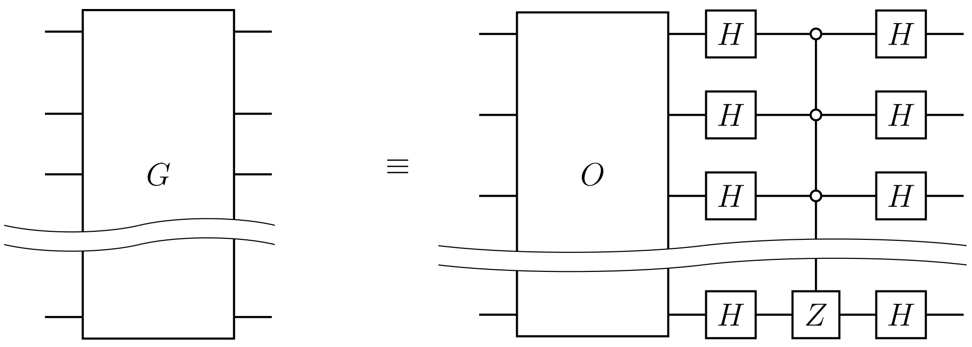


Figure 1.6: Circuit representation for the Grover operator.

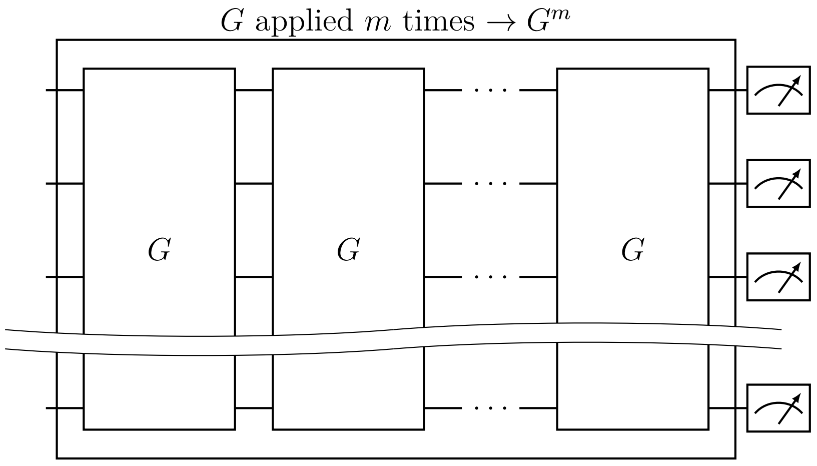


Figure 1.7: Circuit representation for the Grover algorithm.

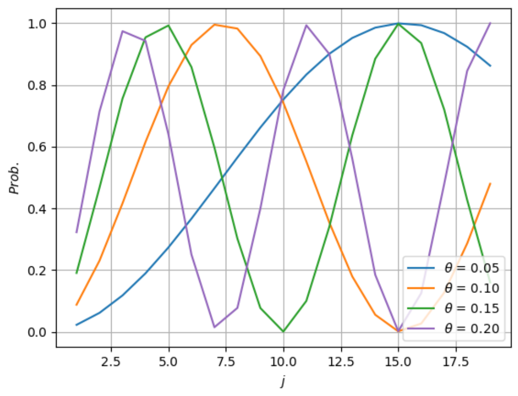


Figure 1.8: Probability of success in finding a desired state with j applications of Grover operator for different values of the number of possible solutions, $\sin^2\theta = t/N$.

$1 \leq t \leq 3N/4$. A maximum for the number of iteration steps, m, and an increasing factor, $1 < \lambda < 4/3$, are employed. First, initialize $m = 1$ and $\lambda = 6/5$ (any value in the allowed range can work). Then, iterate until a solution is found by

1. choosing j at random in $[1, m]$,

2. applying Grover's operator j times starting form $|\psi\rangle$ of Eq. 1.19.

3. Measuring the register, let x be the outcome,
4. if $f(x) = 1$ is met, finish the procedure;
5. otherwise, set $m = \min(\lambda m, \sqrt{N})$ and repeat from step 1.

The cited algorithm finds a solution in expected time $\mathcal{O}\left(\sqrt{N/t}\right)$. In fact, they find an upper bound for the expected number of iterations of $\frac{9}{4}\frac{N}{\sqrt{(N-t)t}} \approx \frac{9}{4}\sqrt{\frac{N}{t}}$. This is only about 4 times greater than the expected number of iterations if t were known beforehand. That is, this algorithm is a suitable alternative to the searching algorithm for a known number of solutions.

1.3.4 Finding the minimum

C. Dürr and P. Høyer proposed an algorithm to find the minimum of an unsorted table [10] , based on the previous algorithm. We desire to find the minimum in a table T allowing indexes from 0 to $N-1$.

Initially a threshold index $0 \leq x_{th} \leq N-1$ is chosen uniformly. Then, the following steps are repeated more than $22.5\sqrt{N} + 1.4\log^2 N$ times:

1. Initialize the quantum state to $\sum_j \frac{1}{\sqrt{N}} |j\rangle |x_{th}\rangle$.
2. Apply the searching algorithm in Ref. [9] and described in the previous subsection, with an oracle which indicates whether $T[j] < T[y]$.
3. Observe the first register, let x' be the outcome.
4. If $T[x'] < T[x_{th}]$, then set x_{th} to x'.

This procedure takes advantage of the fact that in step (2) a suitable value, $T[x'] < T[x_{th}]$, is obtained with high probability, thus lowering the value for x_{th} in most of the iterations.

Chapter 2

Introduction to Genetic Algorithms

On March 22, 2006 the first mission with computer evolved antennas launched in NASA's Space Technology 5 mission [11], even though it was in the 1960's that J. Holland invented genetic algorithms [12].This algorithms were inspired by evolution and genetics. The computer evolved antennas are an example of the engineering optimisation problems that can be solved with genetic algorithms.

In this chapter, which is a review of chapter 1 of Ref. [1] and [12], we will present the grounds of genetic algorithms (GAs). We will see some variants of the algorithms and the parameters that must be chosen when applying them. First we will review the most common structure for the paradigmatic algorithm. After that we will see some examples and different parameters and implementations will be discussed.

The Python 3 programs I developed to get the results shown for the examples in this chapter are available in the repository of the University of the Basque Country (ADDI).

2.1 What is a genetic algorithm?

A genetic algorithm is a procedure to search for an optimum solution to a problem, making use of numerical operations that resemble natural evolution. There are many ways to implement such algorithms, we will focus on a typical genetic algorithm [1, 12]. In any case, to address a problem with these algorithms a method is required to encode each possible solution, as well as a fitness criteria that tells us how well that solution deals with the problem at hand.

(1) First a number of random solutions is generated. These are *chromosomes* which form the initial population. (2) Then, the *fitness* criteria is evaluated for each chromosome of the population. (3) According to those values, the members of the population are *selected* by pairs randomly, assigning higher probabilities to higher fitness solutions. With some probability p_c a *crossover* operation is applied to those members, combining their encoded information. Each bit of the new pair of chromosomes is *mutated* with probability p_m. Step (3) is repeated until a complete set of offspring is generated. Iterations of steps (2) and (3) are called *generations* and they are concatenated until an ending criteria is met. This typical algorithm is depicted in the flow diagram in Fig. 2.1.

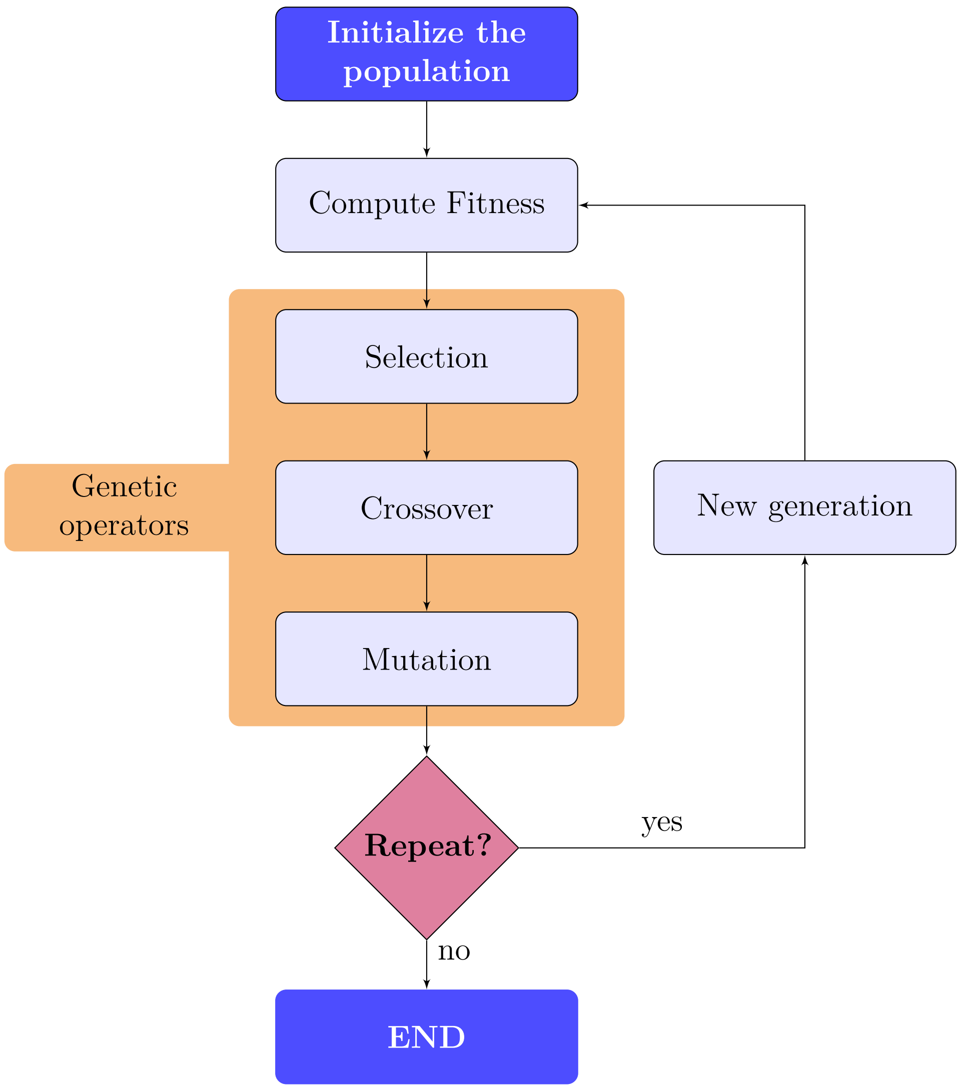


Figure 2.1: Flow diagram representation of a simple genetic algorithm.

2.1.1 Search space and fitness landscape

In biology, the term referring to the genetic makeup of an organism is genotype, while the features that an organism shows are denoted by phenotype. The latter is determined by the genotype and the environment in which the living being grows. In the terms of GAs, the genotype is the particular code that is assigned to a certain member of a population. The phenotype is related with the fitness of that member, which may also depend on other external factors, such as the genotype of other individuals. Choosing an encoding method for the individuals is an important task, as we will see in the examples below.

By extension, we introduce two main concepts in GAs: Search space and fitness landscape. The search space is the set of solutions that could possibly be encoded. The fitness landscape is the set of possible fitness that can be achieved with each individual of the search space. On the grounds of these concepts, GAs try to find an individual in the search space that has a high value in the fitness landscape. GAs implicitly assume there is a relation between the values encoded in the search space and the corresponding fitness value. This relation is leveraged in the selection, crossover and mutation process.

2.1.2 The genetic operators

Selection. This is the process of choosing pairs of chromosomes from the population. It applies the evolving pressure to the population, selecting with higher probability the fitter chromosomes. A typical method is the fitness-proportionate selection, which assigns a selection probability proportional to the fitness. There are other methods, such as elitism that only selects from the best chromosomes.

The choice of the selection method can affect the performance of the algorithm. If there is too much pressure, the best chromosomes rapidly dominate the population and the exploration of the search space will be left to mutation. However, if selection pressure is too small, the population will have no incentive to improve. The proper selection method is problem dependent and balances the exploration with the improving motivation. In the terms of function optimisation methods, a good selection pressure searches for the best maximum without getting stuck too early in local maxima.

Crossover. This operator takes two parent chromosomes in order to spawn two new ones. Single point crossover consists on choosing a random index i in the chromosome. Then, one of the offspring members is generated by taking the first part, from 0 to i, of the first chromosome combined with the second part, from i to the end, of the second chromosome. The other offspring chromosome combines the remaining parts of the parents. Some other methods use two crossover indexes or choose the index using a probability distribution.

Mutation. This is the only operator that adds new information to the evolving process. Usually, each bit of the chromosome is flipped with a probability p_m, which leads the population towards the exploration of the search space. If the mutation does lead to an improvement, it will probably survive and make the average fitness better. Nevertheless, if the mutation rate is too big that may jeopardize the convergence of the algorithm.

2.2 Example 1: Optimisation of a two-variable function

Although the optimisation of an analytic function is not the best example of the performance of a genetic algorithm, it is instructive. Suppose we want to find an optimum point in the surface in Fig. 2.2. In this example, we can easily identify the fitness landscape (height of the surface in each point) and the search space (the x and y coordinates that can be chosen).

More precisely, the search space will be the binary representation of x and y. I use 32 bit numbers to run the test and I use a binary representation so that the number of combinations per area is uniform. This is a good chance to understand the importance of the encoding. If I choose a 32-bit floating-point representation with one bit for the sign, 8 bits for the exponent and 23 bits for the fraction, I would have two main problems. On the one hand, the interesting part of the surface is limited in the x and y axis, so there would be a lot of incorrect chromosomes (out of that area) if I took floating-point representation, as it is designed to have a wide range. On the other hand,

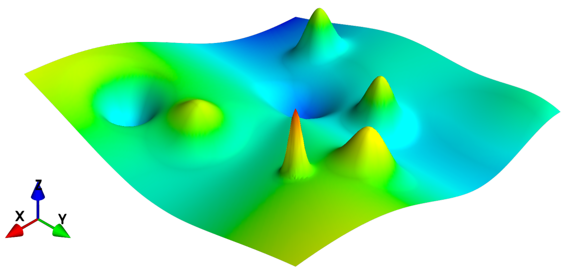


Figure 2.2: Surface for optimisation.

that representation has a highly non-uniform distribution of the points it can represent (there are more condensed near very small numbers), so as a matter of probability our algorithm would be strongly biased towards small numbers. Once I ensured a uniform distribution for the point that could be encoded in each axis, I will have 32 bits for x followed by 32 bits for y[1].

For this example, I choose $N = 20$ chromosomes in the population, fitness proportional selection, single point crossover with probability $p_c = 0.7$ and bit-mutation rate $p_m = 0.001$. These are typical values [1], although they may work poorly in other problems.

We can follow the evolution of the population from different viewpoints. I will follow a similar approach to K. A. De Jong in *Genetic Algorithms Are NOT Function Optimizers* [13]. There he stated three different perspectives and emphasized that none of them captures the whole dynamics of GAs.

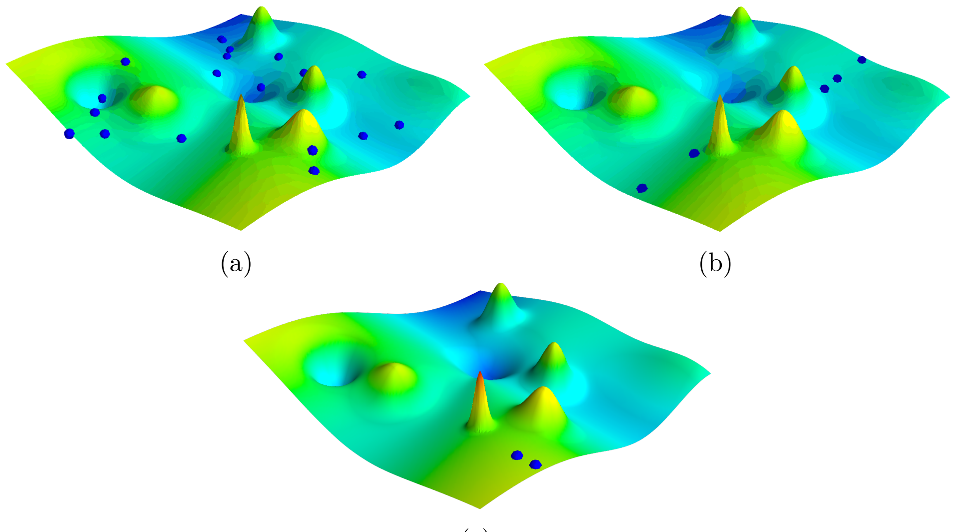


Figure 2.3: Evolution of the points in the population, for generation 1 in (a) , 50 in (b) and 190 in (c).

We can see the evolution of the phenotype, that is, the coordinates of each point of the population. In Fig. 2.3, we can see the population for different generations. Initially (Fig. 2.3a), we have 20 points distributed at random in the surface. For 50 generations (Fig. 2.3b), we can see that all of the points share the same y value and, as we can only distinguish 5 different points, all of the 20 points are allocated in 5 groups. For the case

[1] x's bits with y's could be intercalated if a strong correlation between them was expected.

of 190 generations we can only distinguish two points, which means that in that stage there are only two types of chromosomes. That will not be true forever, as mutation will introduce variations sooner or latter.

From this viewpoint we can clearly see one of the features of GAs. The tendency of these evolutionary algorithms is to converge to a resilient maximum. In this case that is not the global maximum. Crossover and mutation transform the population in a way that makes the convergence to sharp maxima difficult.

We can also see the performance of GAs as function optimisers. This approach may be useful to see how selection effectively leads towards the convergence in a high value in the fitness landscape. Nevertheless, it is not a recommended approach as it hides many features of the evolution. An example for the same process in Fig. 2.3 is shown in Fig. 2.4. In the latter figure, we cannot understand the disappearance of the peaks near generation 20 without keeping other pictures of the evolution in mind. We can also see that mutation affects the mean value, sometimes bringing in better chromosomes or those fading out in the next generation due to selection.

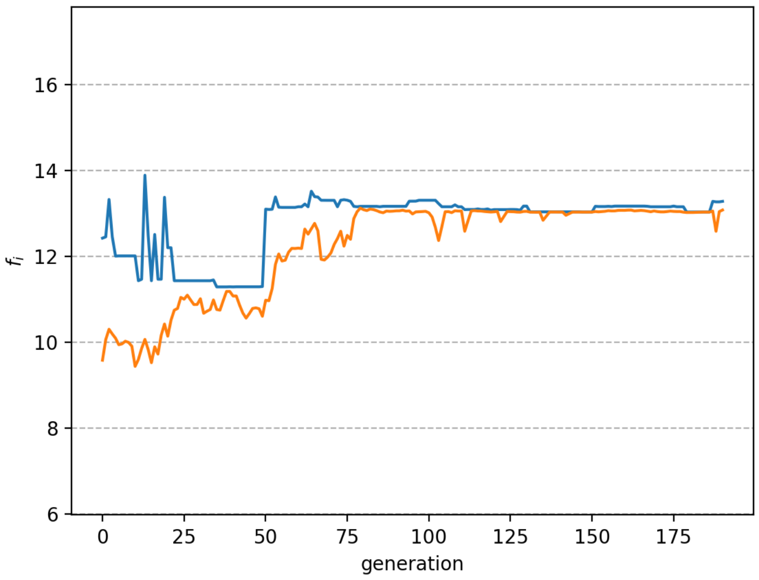
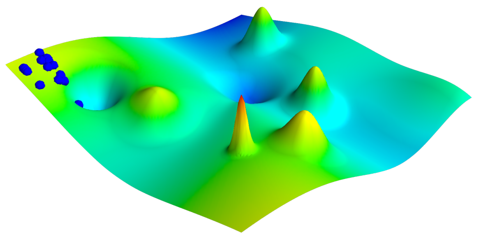


Figure 2.4: Evolution of the mean fitness (orange) and the maximum fitness (blue).

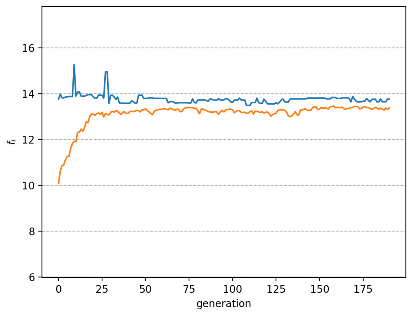
The last viewpoint would consist in analysing the raw genome of the chromosomes. This would imply the detection of the repeated structures in the bit strings and the detection of bit structures which are promoted along the evolution. This analysis is particularly interesting when we are not only looking for a solution, but also trying to learn what makes that solution good. As it would require deeper explanations I forward the interested reader to Ref. [13].

2.2.1 The choice of the parameters

For the above example $N = 20$, $p_c = 0.7$ and $p_m = 0.001$ worked well. Now we will see what happens when some of those parameters are changed. Clearly, by increasing the number of chromosomes in the population N, better results can are obtained. The greater the population number is, the better the ability to explore it will provide. Big populations ensure a constant search in new areas of the search space, while keeping the stability for the overall population. We can see in Fig. 2.5 what is obtained if 100 chromosomes are used, while keeping other parameters fixed. The genetic diversity is kept for longer as we can see in Fig. 2.5a and a smother evolution of the mean fitness is obtained, Fig. 2.5b. Once again, the maximum obtained is a resilient one.



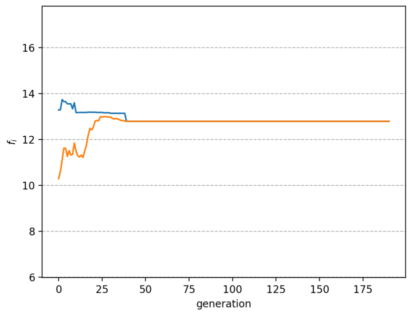
(a)



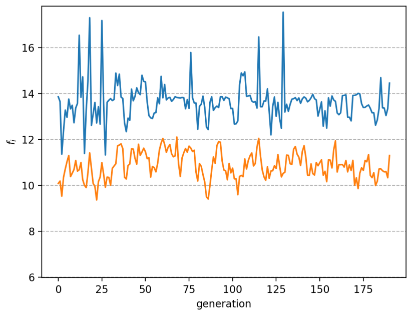
(b)

Figure 2.5: GA performance with population number $N = 100$ (a) point distribution in the surface at generation 190 and (b) evolution of the maximum fitness (blue) and mean fitness (orange) .

The effect of changing the bit-mutation ratio p_m is complex. In Fig. 2.4, we portrayed the performance for $p_m = 0.001$, that is, 1 bit out of 1000 was mutated in each generation. In Fig. 2.6a, we see the effect of turning off mutation, $p_m = 0.000$. Once a chromosome prevails in the population there is no way new information can go into the following generations, in this case evolution stops radically near generation 40. At the other extreme, there is the result obtained in Fig. 2.6b where mutation was increased too much, $p_m = 0.1000$, which means that 1 bit out of 10 was mutated. Such a high value hinders the population from converging. We can see that many good solutions are found, but the evolution cannot prosper near them.



(a)



(b)

Figure 2.6: Evolution of the maximum fitness (blue) and mean fitness (orange) with different mutation ratios (a) $p_m = 0.000$ and (b) $p_m = 0.100$.

The change in the crossover parameter cannot be seen in a couple of images. Thanks to crossover and selection we improve the overall performance of the population in each generation, while keeping similarity in the chromosomes. They are similar because they are likely to come from similar parents and have inherited some bit structures within. In the main example shown in Fig. 2.3, we see that in generation 50 most of the chromosomes have a common axis coordinate, which means that at least they share 32 bits. Turning down crossover, $p_c = 0.0$, means that only selection happens, thus we could have completely different chromosomes in the beginning and then smaller groups of families. In the families we would have very small variations (only due to mutation) and between families we would have completely different bit strings. Within a few generations one of the families will domain, reducing the overall searching ability. Crossover should not be always applied either, $p_c = 1.0$, so as to promote the survival of the best chromosomes.

2.3 Example 2: Evolution of strategies to win games

In a game, players have to take decisions in order to maximize their payoff. We will go through an example to see how decisions can be encoded and their payoff be used as the fitness function. The example I chose is based on the Prisoner's Dilemma described in Section 1.9 in Ref. [1].

Alice and Bob are criminals in a band that in their last theft were caught by the police. The officer has enough evidence to send them to prison, but with their confession a higher sentence can be obtained. To acquire this the officer offers them, separately, a deal: "If you keep quiet we can send you 2 years into prison, while if you confess you can go free and only your friend will be sentenced, for 5 years. If both of you confess, you will go 3 years, collaboration reduces the sentence". They are not allowed to speak to each other while they make a decision. Is it a matter of trust? If each of them confesses, maximizing their personal payoff, they end up in a situation worse than if both of them trusted each other.

The situation can be described in terms of positive payoff 5 - *years_in_prison*, obtaining the payoff Table 2.1. Obviously, if this game is repeated once and again the most profitable strategy is to agree to cooperate. But what if they do not rely on the other player? I will try to evolve the best strategy with a genetic algorithm.

		Player B	
		C	B
Player A	C	3, 3	0, 5
	B	5, 0	2, 2

Table 2.1: Payoff table for the Prisoner's Dilemma. C stands for cooperate and B for betray.

First of all, I have to choose the encoding for the strategies. The players will be provided with the memory code of what they and the other player did in the previous three games. Then, they will make a decision according to that memory code. Cooperate will be encoded with a 1 and betray with a 0. Consequently, a *strategy* is a function of the six binary values in the memory code which returns whether to cooperate or to betray. Strategies will be encoded in 70-bit chromosomes. 64 bits tell what to do according to the memory code and the other 6 hold a memory code to be used if no previous game is registered. For example, if strategy A faces strategy B both will be given what they did in the previous three games, e.g. ’01 10 10’ for A and ’10 01 01’ for B. The corresponding integer number is 26 for A and 37 for B, thus each strategy will look at that position in the 64-bit string and decide to cooperate if a 1 is encoded or to betray if a 0 is encoded. Finally, each strategy will get a score according to Table 2.1.

I will have 10 initial chromosomes which will play 100 times against each other and their fitness will be their overall score. For this problem, I run for 50 generations and perform 10 different runs.

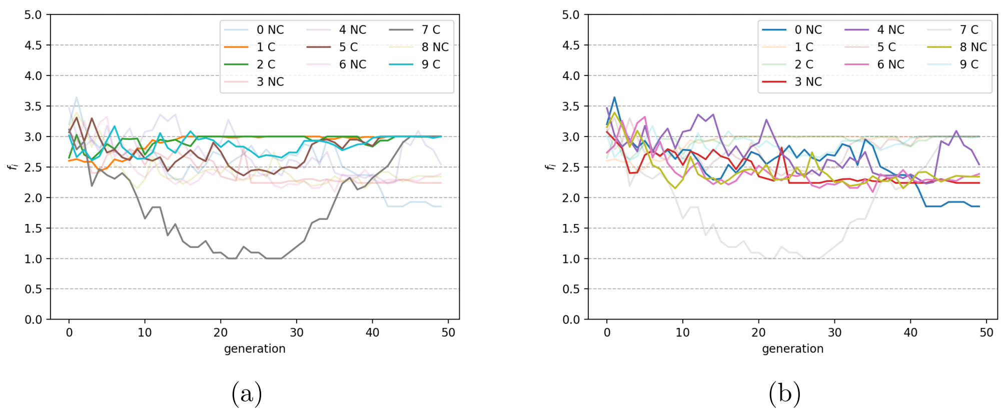


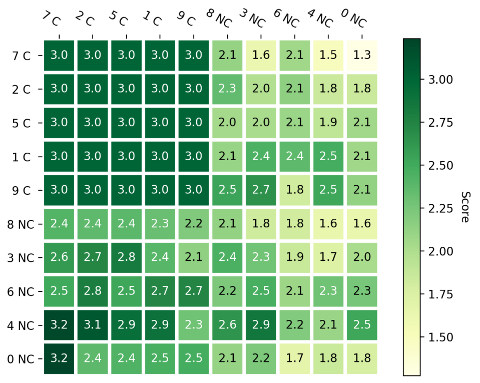
Figure 2.7: Evolution of the fitness for the best strategy for the Prisoner’s Dilemma. Those marked as C highlighted in figure (a) are the trials that evolve strategies that score 3, related with cooperation. In the other hand, those marked as NC highlighted in (b) are the strategies that do not converge to 3 in 50 generations.

In Fig. 2.7a, we see that some trials converged to a score of 3. We can assume this is due to a cooperating pattern in the population. This does not mean that they always cooperate (a strategy full of ones), but that they always cooperate with each other. We can think of it as if they cooperated in the first 3 games and got memory codes of cooperation, 111111, then to keep cooperating they only need to have a 1 in index 63. From this figures we cannot tell if they would act similarly against other kind of strategies. In Fig. 2.7b, we see that other strategies did not converge to any value. Within those populations sometimes they will cooperate and others they will not.

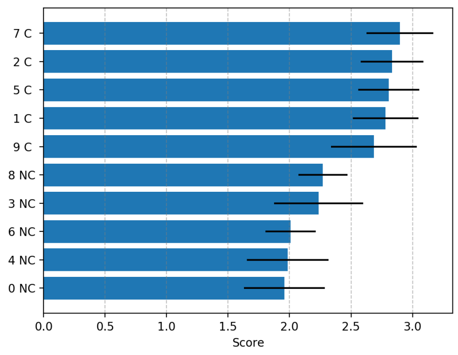
The stability of cooperating strategies can only be understood if we keep in mind that 100 games are played against each member of the population, i.e. strategy. If a strategy got a high score based on betrayal, it will get more chances to have more offspring, which will consequently be prompt to betray others. That quickly tends to a lower overall score. In this sense cooperating strategies are more stable when they face players that follow similar strategies.

In order to get a further insight on how this strategies work, I performed a tournament in which each final population faced all the others. In Fig. 2.8a, I show the score that each strategy has obtained against the others. The strategies are ordered based on the overall performance that can be seen in the ranking of Fig. 2.8b.

Figure 2.8a shows that chromosomes behaving similarly when playing within their own population act very differently against different populations. This accounts for the different results obtained from the cooperating strategies. Some of them, as the 7th



(a)



(b)

Figure 2.8: Mean score of each evolved strategy for the Prisoner's Dilemma (a) score of columns against rows and (b) average of the score against all the strategies, the black bars reflect the deviation from that average value.

against the 4th or 0th, clearly betrayed sometimes in order to get a higher than 3 profit. Anyway, cooperating populations tend to cooperate with each other, obtaining higher overall profit in Fig. 2.8b.

2.4 Applications

GAs have been used in a wide variety of disciplines, such as natural sciences, economics, engineering, social sciences... As an example, evolutionary algorithms are frequently used in electronics design [14].

In the introduction of chapter I mentioned the antenna created for NASA's ST5 mission [11]. For this mission, a contractor was hired to design an antenna that could met the requirements for the mission efficiently. An evolutionary algorithm was also developed to obtain a suitable antenna. The success of the evolved antenna was unquestionable: The energy consumption efficiency was 93% for the evolved antenna versus the 38% obtained by the conventionally designed one. Moreover, the evolution procedure took only 3 person-months of work, even with several performance requirements being changed, compared to 5 person-months of work. The evolved antenna also provided a more uniform coverage.

To sum up, genetic algorithms perform well when the fitness landscape is unfamiliar or complex constraints are considered. They are also useful to simulate evolutionary behaviour. Genetic algorithms are robust algorithms that can easily be implemented to solve almost any problem. Usually, a problem specific algorithm may perform better in getting a global optimum, as in the example of the two dimensional function where an steepest-ascent could find better maxima. A great benefit GAs will always provide is that they find resilient solutions.

Chapter 3

Towards Quantum Genetic Algorithms

Quantum computation and genetic algorithms have been combined in recent years. On the one hand, some researchers have used genetic algorithms to develop quantum circuits. On the other hand, a genetic algorithm with an enhanced selection procedure has already been developed. That selection subroutine was enhanced employing quantum searching algorithms. In Section 3.1, I explain that algorithm and propose a variation for the selection procedure.

In Section 3.2, I develop a quantum genetic algorithm. This was obtained combining the quantum version of each step in the genetic algorithm. A first test is performed and a preliminary performance measurement is obtained. Although further study is needed, the structure of the proposed algorithm is developed in separated registers (qubit arrays) and could be distributed. This algorithm could be a suitable example to explore distributed quantum computation.

The Python 3 programs I developed to obtain the results shown in this chapter are available in the repository of the University of the Basque Country (ADDI).

3.1 Quantum enhanced selection

In Ref. [15], a quantum genetic optimisation method is proposed. Recalling the genetic operators in Section 2.1.2, in this work the selection procedure is performed by applying a truncated version of the Dürr-Høyer quantum optimisation routine, explained in Section 1.3. My aim in this section is to explore other quantum selection methods that could also be used to enhance the classical genetic algorithm.

3.1.1 Selection with truncated Dür-Høyer optimisation

The quantum selection procedure stated in Ref. [15] needs $\log_2 N$ qubits to represent the whole search space, $x \in \{0, 1, ..., N-1\}$, and starts assuming a reference individual x_{th}, which implies a threshold value $F(x_{th})$. Each value is stored in different qubit strings called *registers*. The value x_{th} can be retrieved from the previous generation. Then, Dürr-Høyer optimisation is applied, explained in Section 1.3.4, with iterations truncated to a value n_h and the oracle shifted to indicate $T[j] > T[y]$. This results in

1. Initializing the quantum state to $\sum_j \frac{1}{\sqrt{N}} |j\rangle |x_{th}\rangle$.
2. Applying the searching algorithm in Ref. [9], with an oracle that marks $T[j] > T[y]$.
3. Observing the first register, let x' be the outcome,
4. if $T[x'] > T[x_{th}]$, then set x_{th} to x'.
5. After n_h iterations, return x_{th}.

This implies that the selected individual will have high fitness. This improvement reduces the cost of the selection step, while keeping a wide range of selection possibilities. In the classical selection procedure $\mathcal{O}(N \log N)$ calls to the oracle are needed, to get the individuals sorted and achieve a good chance to find a high value. Meanwhile, in the quantum selection procedure the number of oracle calls strictly depends on the parameter n_h, which does not vary with N.

3.1.2 Selection with multiply compared amplification

As a variation to the previous quantum selection procedure (step 2), I have developed a multiply compared amplification based selection. *Amplitude amplification* is the technique of enhancing the amplitude of some desired states by means of quantum gates. *Multiply compared amplification* highlights the fact that several threshold values are used at the same time. We will use index notation, so that we get generalized solutions and so that we can implement and study case examples. In that sense, the Grover operator defined in Eq. 1.20 performs,

$$\langle i|G|j\rangle = G_{ij} = \left(\frac{2}{N} - \delta_{ij}\right) O(j), \quad \text{with } O(j) = \begin{cases} -1 & j \in A, \\ 1 & j \in B. \end{cases} \tag{3.1}$$

A being the subspace of desired solutions, $j \in A$ if $F(j) > F(x_{th})$, and B the complementary subspace, where F is the fitness criteria. This operator is used to amplify the amplitude of the states that satisfy $j \in A$. To perform this we can take a second register that will work as a threshold register so that,

$$\langle i, x|G|j, x_{th}\rangle = \left(\frac{2}{N} - \delta_{ij}\right) \delta_{x,x_{th}} O(j, x_{th}), \quad \text{with } O(j, x_{th}) = \begin{cases} -1 & F(j) > F(x_{th}), \\ 1 & F(j) \leq F(x_{th}). \end{cases}. \tag{3.2}$$

Note that as the operator O can be checked to be unitary G is also unitary. The operator G transforms a given state $|\psi\rangle |x_{th}\rangle = \sum_j a_j |j\rangle |x_{th}\rangle$ to $|\psi'\rangle |x_{th}\rangle = \sum_i a'_i |i\rangle |x_{th}\rangle$, where

$$a'_i = \sum_j \left(\frac{2}{N} - \delta_{ij}\right) O(j, x_{th}) a_j. \tag{3.3}$$

This operator is used in the above mentioned quantum selection procedure, initializing the state to $a_j = 1/\sqrt{N}$. As long as the chosen threshold is above the median value of the fitness, the state resulting from $G|\psi\rangle |x_{th}\rangle$ returns an individual with higher fitness than a threshold with high probability. The probability distribution governing the values that can be obtained is a step function, small for the lower than threshold individuals and high for the higher than threshold individuals. I propose a modification close to achieving a probability distribution monotonically increasing with the fitness of the returned individual.

The desired behaviour can be achieved replacing $|x_{th}\rangle$ by a superposition in the computational basis $\sum_\nu b_\nu |\nu\rangle$. Then, the initial state reads $|\phi\rangle = \sum_j a_j |j\rangle \sum_\nu b_\nu |\nu\rangle$, and the transformation achieved applying the operator in Eq. 3.2 is,

$$G|\phi\rangle = \sum_{i,\nu} b_\nu \left\{ \sum_j \left(\frac{2}{N} - \delta_{ij} \right) O(j,\nu) a_j \right\} |i\rangle |\nu\rangle \, . \tag{3.4}$$

Appropriately tuning b_ν we can achieve the desired distribution. We name the first register *searching register* encoding the searching state, and the second one *threshold register* encoding the threshold state.

Numerical examples

I will also assume an initial uniform superposition in the searching register, that is $a_j = 1/\sqrt{N}$. I will focus on the analysis of the output probability of each state $|i\rangle$ after a single application of G. Thus the distribution to be tuned is

$$P(i) = \sum_\nu \frac{|b_\nu|^2}{N} \left(2 - O(i,\nu) - 4\frac{t_\nu}{N} \right)^2 , \tag{3.5}$$

where t_ν is the *order number* denoting the number of j values that satisfy $O(j,\nu) = -1$, or $F(j) > F(\nu)$. Note that the searching state space and the threshold search space are dimensionally equivalent.

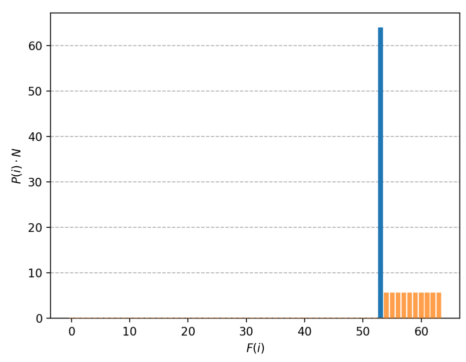
To begin with, we will consider a fitness proportional to the integer expressed in the base-2 numeral system. I show in Fig. 3.1 the output probability I obtained for different initial distributions of $|b_\nu|^2$.

In Fig. 3.1a and 3.1b I performed the amplification setting the threshold register to a single value. Clearly, the higher we choose the threshold, the better the result we will obtain. Moreover, as we can see in Fig. 3.1b, going below the center value in the threshold amplifies lower values instead of higher ones.

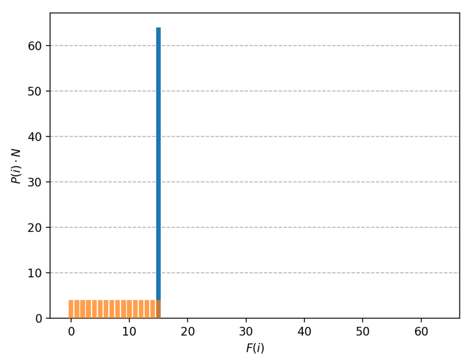
In Fig. 3.1c, I employed a uniform distribution in the threshold register. The symmetry in Eq. 3.5 implies that amplifying the highest fitness individuals involves amplifying the lowest ones too. This symmetry can be broken with an asymmetric distribution in the threshold register, as shown in Fig. 3.1d. But, if we were able to get such a distribution our aim would be easily achieved! That is not the case. For this examples, I used an small toy model and every calculation can be tailored to our desires with classical computers. Therefore, a quantum procedure in order to amplify the greater values without boosting the lower ones must be developed.

Previously amplified threshold

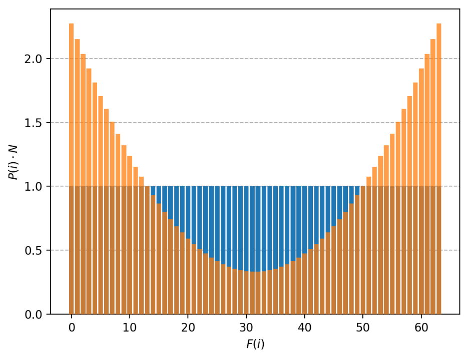
We can easily achieve a good enough initial individual so that its fitness is above the median value. This means we can obtain the case shown in Fig. 3.1a efficiently. Our



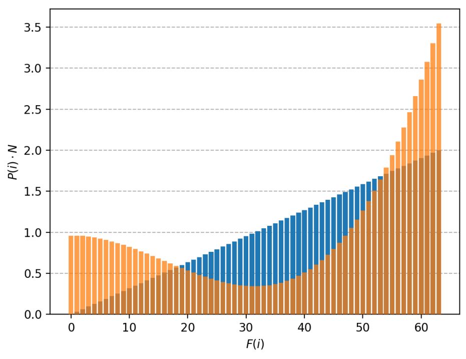
(a)



(b)



(c)



(d)

Figure 3.1: Output probabilities (orange) for different threshold distributions (blue). I show $P(i) \cdot N$ to provide a better insight of the redistribution of probabilities. The search register initially always has $P(i) \cdot N = 1$.

proposal is to first initialize the search register to a single individual (as the blue bar shown in Fig. 3.1a), then, use the *search register* as a threshold for the *threshold register* (previously initialized in a uniform superposition), so that we get a distribution like that I obtained for the orange bars in Fig. 3.1a. Both registers are in separable states, so one can safely reset the search register and get a uniform superposition. Finally, apply the amplification operator using the distribution in the threshold register as the threshold. In practice this result is similar to amplifying the search register with a step distribution in the threshold register. Performing this steps, I obtained the result shown in Fig. 3.2a.

In this procedure, each state i is only distinguishable by its order number t_i. Individuals with the same fitness have the same order number and are affected in the same way by the algorithm. This implies that any bijective fitness function will obtain the same results. If the number of repeated fitness values is small, the result is similar. As examples we can see results in Fig. 3.2.

The efficiency of this procedure depends on the order number of the first threshold, t_z. As I show in Fig. 3.1a and 3.1b, if the initial threshold is too bad the lower states are amplified. Similarly, if a threshold among the best is employed the step obtained will be sharper and will enhance the higher values better. In Fig. 3.2, the threshold value was chosen to fulfil $t_z/N \approx 0.25$. Those figures show the proportional increasing amplification of the computational basis states that exceed the threshold fitness, which is imposed by the threshold index.

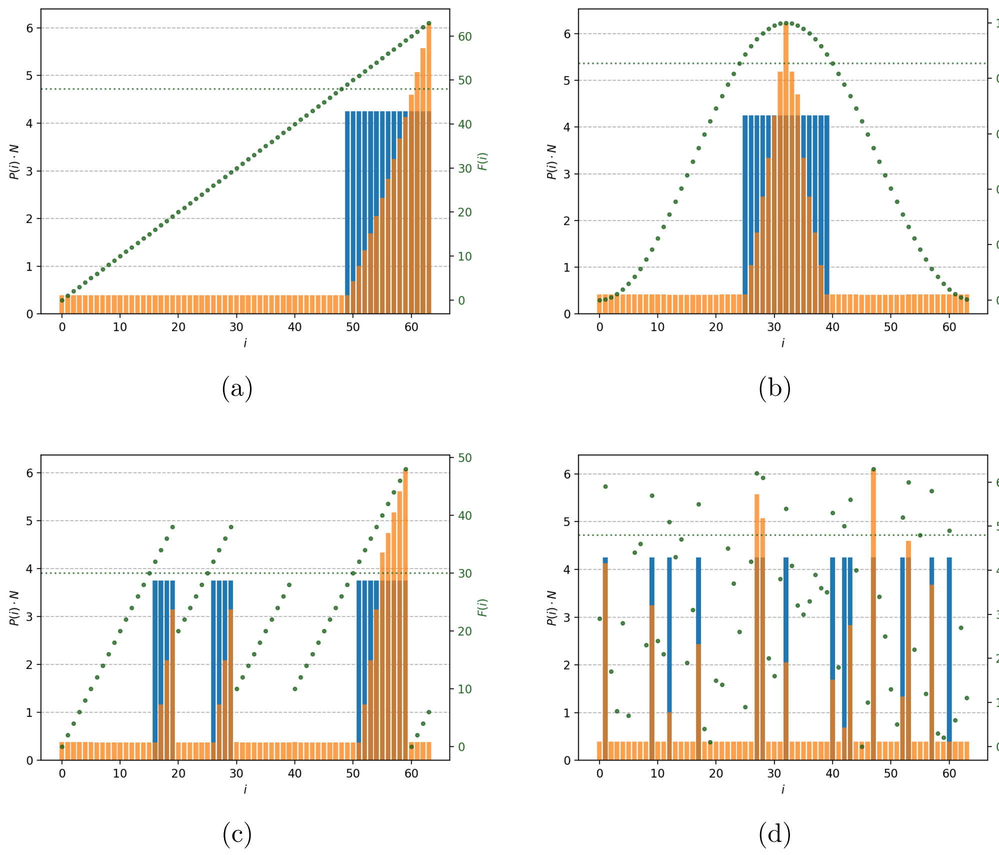


Figure 3.2: Relative output probabilities (orange) for different fitness functions (green). The threshold distributions (blue) are obtained with single threshold amplification, using a threshold individual with order number $t_z/N \approx 0.25$. The functions $F(i)$ are (a) i, (b) $\sin^2(\pi i/64)$, (c) $i \mod 20 + i \mod 30$ and (d) unsorted i.

Although I obtained an increasing probability, associated with the fitness of the individuals, I paid the cost of amplifying the lower fitness states too. In fact, regarding the probability to get a state strictly above the threshold, the procedure I proposed gets worse results. Regarding a bijective function with single threshold I obtain a probability to measure above the threshold index z of

$$\sum_{i:F(i)>F(z)} P_{sca}(i) = \sum_{i:F(i)>F(z)} \frac{1}{N}\left(3 - 4\frac{t_z}{N}\right)^2 \tag{3.6}$$

$$\sum_{i:F(i)>F(z)} P_{sca}(i) = x_z\left(3 - 4x_z\right)^2, \tag{3.7}$$

obtained from Eq. 3.5 setting $b_\nu = \delta_{\nu z}$ and summing over all possible solutions, also defining $x_z = t_z/N$ and noting that t_z goes from 0 to $N-1$. On the other hand, for the multiply compared amplification we have to divide the summation over ν in three subspaces: A for ν satisfying $F(\nu) \leq F(z)$ with $N - t_z$ terms, B for ν satisfying $F(z) < F(\nu) \leq F(i)$ with $t_z - t_i$ terms and C for ν satisfying $F(\nu) > F(i)$ with t_i

terms,

$$P_{mca}(i : F(i) > F(z)) = \sum_{\nu \in A} \frac{1}{N^2}(1-4x_z)^2(3-4x_\nu)^2 + \\ \sum_{\nu \in B} \frac{1}{N^2}(3-4x_z)^2(3-4x_\nu)^2 + \sum_{\nu \in C} \frac{1}{N^2}(3-4x_z)^2(1-4x_\nu)^2. \quad (3.8)$$

Which summing over all i satisfying $F(i) > F(z)$ and approximating to $N \to \infty$ but x_z finite,

$$\sum_{i:F(i)>F(z)} P_{mca}(i) \approx \frac{x_z}{3}\left(7 + 52x_z - 248x_z^2 + 320x_z^3 - 128x_z^4\right). \quad (3.9)$$

Comparing both in the range $0 \le x_z \le 1$ in the Fig. 3.3 we clearly see that for the interesting values, $x_z < 0.5$, single compared amplification gets better chances to find a higher fitness index.

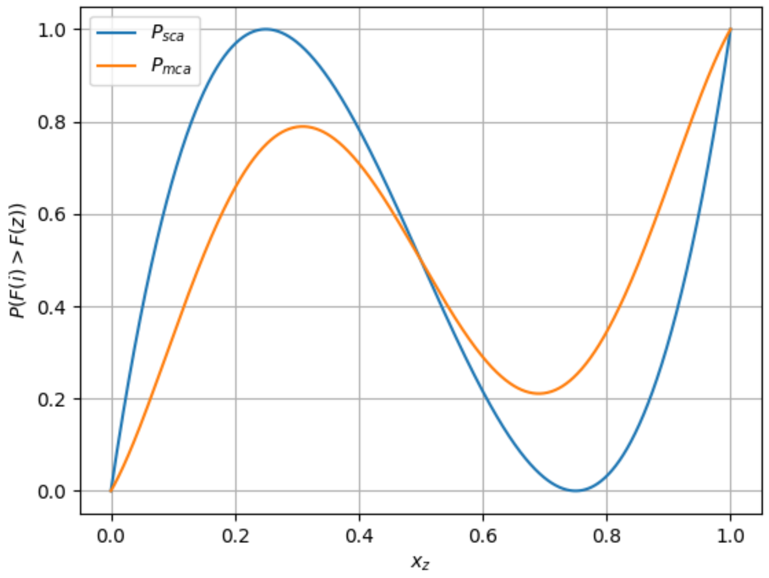


Figure 3.3: Probability to get a better than threshold index with single compared amplification, P_{sca}, and multiply compared amplification, P_{mca}, as a function of the thresholds order number, $x_z = t_z/N$.

Moreover, our procedure cannot be adapted to fit into something like Dürr-Høyer minimum finding algorithm, as it does not strictly match the operation of a Grover operator. Nevertheless, that was not our intention. Genetic algorithms do not seek the straight way up. Usually, we are also interested in obtaining low fitness individuals, in case their genetic codification can add some interesting information to the population.

3.2 Completely quantum version

3.2.1 Case problem and general procedure

The problem studied will be the task of finding the six bit string corresponding to the integer number that maximizes the fitness function

$$f(x) = 2^{10} - (2^5 - x)^2 = 1024 - (32 - x)^2. \tag{3.10}$$

Analytically, this is found to be the central value $x = 32$, which is the string '100000', with fitness 1024. The population number is chosen to be 8, so the population will be encoded in an array of 8 registers with 6 qubits each. All the registers form the quantum system, so a quantum state in the computational basis corresponds to a population in the genetic algorithm. The procedure that I will follow is:

1. Initialize the population, assigning a quantum state.
2. Sort the registers so that the best 4 individuals are in the first 4 registers.
3. Clear the last 4 registers, so that only the best individuals remain.
4. Pseudo-clone each individual in the first 4 registers to the empty ones.
5. Perform a crossover, swapping the last 3 qubits of register 5 with 6 and 7 with 8.
6. Apply a mutation unitary with probability p_m in each qubit.
7. If the ending criteria is met measure the registers, if not go to step 2.

3.2.2 Sorting subroutine

For this purpose we need a procedure that gets 8 registers in an unknown quantum state and sorts them. Sorting will be well defined for the states that represent a classical population, so the procedure will be developed in that basis and then it will be shown that the procedure also works with other states.

For sorting I will define a sorting oracle, O_S, as was shown for the Grover algorithm in Section 1.3. This will be a unitary that taking two registers applies a X gate in an ancilla qubit if they are unsorted and identity if they are sorted. That is,

$$O_S |x\rangle |y\rangle |0\rangle = \begin{cases} |x\rangle |y\rangle |0\rangle & \text{if } f(x) \leq f(y), \\ |x\rangle |y\rangle |1\rangle & \text{if } f(x) < f(y). \end{cases} \tag{3.11}$$

I also require a gate that performs a control swap of two registers,

$$C_{\text{SWAP}} |x\rangle |y\rangle |c\rangle = \begin{cases} |x\rangle |y\rangle |c\rangle & \text{if } c = 0, \\ |y\rangle |x\rangle |c\rangle & \text{if } c = 1. \end{cases} \tag{3.12}$$

In both Eq. 3.11 and 3.12 $|x\rangle |y\rangle |c\rangle$ was assumed to be a separable state and x and y to represent a state of the computational basis. Nevertheless, these definitions are enough to define the operation of the quantum gate in an arbitrary state.

These gates allow us to perform the quantum version of Bubble Sort algorithm. For the sake of simplicity I will explain the method for 8 registers and assume I have 4 ancillas initialized at $|0\rangle$, while the generalization is simple.

1. Apply the sorting oracle, O_S, to the register pairs marking the associated ancilla $(r_1, r_2) \rightarrow a_1$, $(r_3, r_4) \rightarrow a_2$, $(r_5, r_6) \rightarrow a_3$ and $(r_7, r_8) \rightarrow a_4$.
2. Apply the controlled swap, C_{SWAP}, to the register pairs controlled by the associated ancilla $a_1 \rightarrow (r_1, r_2)$, $a_2 \rightarrow (r_3, r_4)$, $a_3 \rightarrow (r_5, r_6)$ and $a_4 \rightarrow (r_7, r_8)$.
3. Set the ancillas to $|0\rangle$, as explained below.
4. Apply the sorting oracle, O_S, to the register pairs marking the associated ancilla $(r_2, r_3) \rightarrow a_1$, $(r_4, r_5) \rightarrow a_2$ and $(r_6, r_7) \rightarrow a_3$.
5. Apply the controlled swap, C_{SWAP}, to the register pairs controlled by the associated ancilla $a_1 \rightarrow (r_2, r_3)$, $a_2 \rightarrow (r_4, r_5)$ and $a_3 \rightarrow (r_6, r_7)$.
6. Repeat 3 times from 1 to 4 and finally apply steps 1, 2 and 3.

The circuit representation of the sorting algorithm for 4 registers is shown in Fig. 3.4. In order to reset the ancilla qubits to zero a projection through a single shot measurement is needed. The simplest way is to use the Z basis, if a 0 is obtained no action is required, while if a 1 is obtained an X gate should be applied. However, these steps can spoil the superposition in the computational basis. Take the case of $\alpha |\psi_0\rangle |0\rangle + \beta |\psi_1\rangle |1\rangle$, there would be two possible outcomes: $|\psi_0\rangle |0\rangle$ with probability $|\alpha|^2$ and $|\psi_1\rangle |0\rangle$ with probability $|\beta|^2$. To avoid this I will perform the projection in X basis, applying a Hadamard gate before the measurement in Z basis. This transforms the state to,

$$\frac{\alpha}{\sqrt{2}}(|\psi_0\rangle |0\rangle + |\psi_0\rangle |1\rangle) + \frac{\beta}{\sqrt{2}}(|\psi_1\rangle |0\rangle - |\psi_1\rangle |1\rangle). \tag{3.13}$$

Performing a single shot measurement in the ancilla we can obtain 0 or 1, with probability 1/2, and apply X only in the second case. The possible outcome states are $\alpha |\psi_0\rangle |0\rangle + \beta |\psi_1\rangle |0\rangle$, if 0 was obtained, and $\alpha |\psi_0\rangle |0\rangle - \beta |\psi_1\rangle |0\rangle$, if 1 was obtained. I employed this resetting method, as it preserves the statistics of the states in the unobserved qubits.

3.2.3 Crossover subroutine

The crossover subroutine is implemented in 3 steps: clearing the lowest half, clone the best individuals to the lower ones and swap part of the copied individuals. To clear the lowest half I employed the same procedure as to reset the ancillas in the sorting subroutine for registers 5 to 8. Then, the cloning was performed from r_1 to r_5, from r_2 to r_6, from r_3 to r_7 and from r_4 to r_8. Finally, the last three qubits of registers r_5 were swapped with the last three registers of r_6, similarly for r_7 and r_8.

As it was explained in Section 1.2, cloning the whole quantum state of a register to another is not possible in general. Later in that section the partial cloning unitary in Ref. [7] was mentioned. This method perfectly clones the quantum states of the

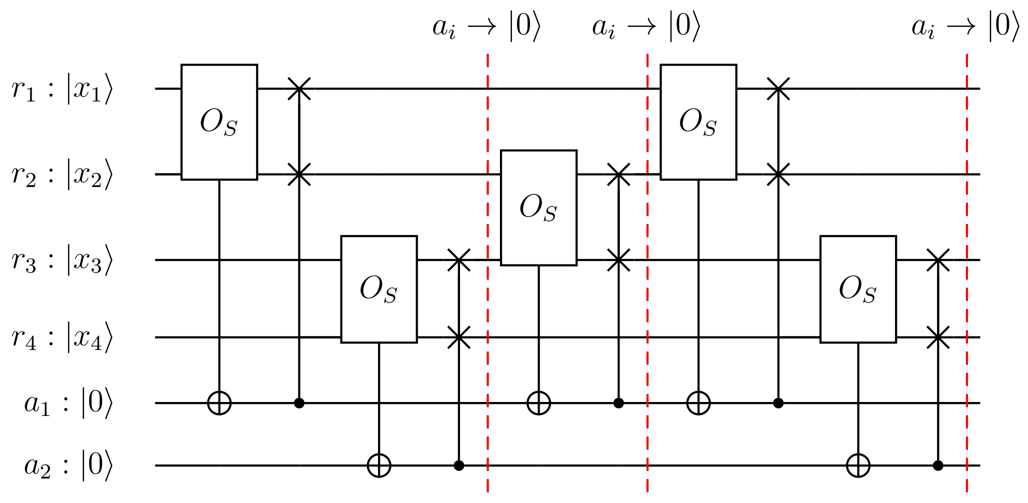


Figure 3.4: Circuit representation for the sorting subroutine with 4 registers.

computational basis, that only contain classical information, and ensures the transfer of the statistics of a chosen observable for any state.

The partial cloning unitary adapted from Ref. [7] satisfies,

$$\langle\psi|M|\psi\rangle = \langle\psi,\psi'|(M\otimes\mathbb{I})|\psi,\psi'\rangle = \langle\psi,\psi'|(\mathbb{I}\otimes M)|\psi,\psi'\rangle\,, \tag{3.14}$$

where $U|\psi\rangle|0\rangle = |\psi,\psi'\rangle$, which is not a separable state in general. This property ensures the correct cloning of the statistics according to observable M, which must be diagonal in the computational basis.

The partial cloning unitary in the computational basis is computed by

$$x_{ni}|k\rangle = \begin{cases} |k+i-1\rangle & \text{if } k \le n-i+1, \\ |k+i-1-n\rangle & \text{if } k > n-i+1, \end{cases} \qquad s_{ni} = |i\rangle\langle i|\,, \tag{3.15}$$

$$U_n = \sum_{i=1}^{n} s_{ni}\otimes x_{ni}, \tag{3.16}$$

where x_{ni} represents a translation group, s_{ni} represents the projection into each subspace and U_n is the partial cloning unitary. This representation allows the partial cloning of a state $|\psi\rangle$ represented in n dimensions. The result is a state space of dimension $n\times n$. The operator U_n is defined in n subspaces of n dimension, each with index i. Within each n dimensional subspace the translation operator x_{ni} is defined. The piecewise definition ensures that $|k\rangle$ and the state it goes to range from $|1\rangle$ to $|n\rangle$ keeping within the initial subspace. With the notation used i ranges from 1 to n. With this definition the pseudo cloning operator can be built as a matrix for computational simulation. For the quantum genetic algorithm I wanted to clone $2^6 = 64$ dimensional states so U_{64} was used operating on the parent and children register.

3.2.4 Mutation

As I mentioned in Section 2.1.2, mutation is essential to introduce new information to the evolution process. In a classical computer there is a single mutation operation to do this, bit flip. The difference is in the criteria used to choose which bit to flip, usually by selecting randomly with probability p_m. The direct analogue in a quantum structure is to decide whether to apply a bit flip, X gate, to each qubit with probability p_m. When the initial state can be described by a single classical state, performing the sorting, crossover and this kind of mutation results in a similar behaviour for the quantum and classical algorithms.

The proposal to avoid this is to substitute X gate with another one that introduces superposition, as the Hadamard gate. In this sense, not only would mutation introduce new information, but it would also add superposition to the state.

3.2.5 Resources needed

As mentioned above, there are $6 \times 8 = 48$ qubits needed to represent the population and 4 needed for the ancillas, for a total of 52 qubits. Moreover, in this preliminary development issues like quantum error correction were not considered, as qubits were assumed to be noiseless. Taking this into account would increase the number of resources needed.

To get quick results the circuit was simulated using classical resources representing operations by means of linear algebra. This involves some restrictions too. As explained in Section 1.1 to represent a state space of n qubits, 2^n dimensional vectors are needed. In this case each operation would imply a $2^{52} \times 2^{52}$ dimensional matrix. For the sake of simplicity restricted cases were studied. For instance, most of the operations involved in the quantum genetic algorithm only apply transitions, meaning that in matrix representation they are sparse matrices which only have a non-zero value in each row and column. This implies that if the initial state can be represented with a few non-zero coefficients the computation may be efficiently run in a standard computer.

To this aim we must ensure that the number of non-zero amplitudes does not grow too much in each generation. This implies restrictions in the initial population chosen and in the mutation method used.

If X is used as a mutation unitary, there is no restriction as it only performs the change $|0\rangle \rightarrow |1\rangle$ and $|1\rangle \rightarrow |0\rangle$, conserving the number of non-zero coefficients in the vector representation. But the H gate can introduce $|0\rangle \rightarrow (|0\rangle + |1\rangle)/\sqrt{2}$ and $|1\rangle \rightarrow (|0\rangle - |1\rangle)/\sqrt{2}$. If we consider a probability p_m to mutate a qubit, 48 qubits and G generations. This results in an expected value of $48Gp_m$ mutations, resulting in a final state with 2^{48Gp_m} non-zero coefficients.

Object oriented programming was used, creating our own classes to take advantage of the particular properties of the matrices. Nevertheless, better computational resources would allow further results, such as bigger mutation probability, larger populations or longer chromosomes.

3.2.6 Results

I compared the results obtained with the quantum genetic algorithm comparing (QGA) to a classical genetic algorithm (CGA). Both created the new generation performing selection by sorting and choosing the first half, cloning them to the lower half and making crossover with these. In the classical version this resembles the logic used for the quantum genetic procedure explained above. The actual difference is that the population state in the quantum genetic algorithm may represent many classical populations at the same time and that mutation was performed with H gate.

To perform a meaningful comparative I first determined a reasonable number of generations. I ran several CGAs with 100 generations and show that by generation 50 80 % of the CGAs run had found an optimum value that did not change in the following 50 generations. I took 50 generations as a reference for the convergence of most of the trials performed.

I performed 200 trials with 50 generations, each trial using the same initial population for CGA and QGA [?]. Their evolution was stored for latter analysis. The parameters chosen were big enough to perform the analysis, but limited because of the QGAs where many mutations occurred which had great computational complexity. The reasons for that are the increase in the non-zero amplitude coefficients due to mutation with a Hadamard gate.

A basic comparative of the performance of each GA requires fixing a threshold value for the fitness of the population in order to see how many trials overcome that limit. More concretely I measured the ratio of final populations that have an individual above a given fitness, for the CGA. For the QGA, I also considered the probability of that population to be measured. That is, define the probability to measure the population i with an individual above the given threshold as $p_t^{(i)}$ for a certain trial t. Consider p_t as the sum of the probabilities of each population meeting the required condition for that trial, $p_t^{(i)}$. Then the measured magnitude was

$$R_q = \frac{1}{T}\sum_t p_t = \sum_t \sum_i p_t^{(i)}. \tag{3.17}$$

We can interpret the result in the CGAs as if p_t could only be 0 or 1,

$$R_c = \frac{1}{T}\sum_t p_t. \tag{3.18}$$

Tolerance (%)	R_q	R_c
0.0	0.42	0.48
2.5	0.78	0.86
5.0	0.91	0.94
10.0	0.95	0.97
15.0	0.97	0.98

(a)

Tolerance (%)	R_q	R_c
0.0	0.23	0.30
2.5	0.27	0.31
5.0	0.35	0.34
10.0	0.90	0.92
15.0	0.96	0.96

(b)

Table 3.1: Ratio of final populations above a given tolerance for QGA, R_q, and CGA, R_c. a) for parabolic fitness function and b) for $x \mod 20 + x \mod 3$.

With this definitions we can see that the result obtained for both are very similar in Table 3.1a. The tolerance was defined as the proportion of individuals in the search space that are higher or equal to a given threshold. I tried also with a different fitness function, $f(x) = x \mod 20 + x \mod 30$ as shown in Fig. 3.2c with green dots. I performed the same measurement and obtained the results in Table 3.1b.

Conclusions

Technological advances developed with genetic algorithms show their ability to find robust solutions to problems under complex constraints. Nevertheless, these algorithms require elaborated selection procedures and an unknown number of iterations. It is reasonable to explore if quantum resources can eventually provide any advantage to genetic algorithms, as it has been the case for other quantum algorithms. Moreover, developing a quantum genetic algorithm can also provide a new insight for quantum computation as they are natural candidates to be useful distributed quantum algorithms.

This work starts describing the grounds of quantum computation. We have review the concepts of qubit, quantum gate and measurement. Then, we briefly explored the state of the art in searching and minimum finding algorithms, recalling their speed-ups. We have also reviewed one of the principal results in quantum information: the no-cloning theorem. It was shown that a quantum state can not be completely copied which leads to the design of approximated cloning procedures for the development of a quantum genetic algorithm.

In the second chapter a survey on some fundamental concepts about genetic algorithms is proposed. I deepen in the building blocks of these algorithms: selection, crossover and mutation. Explained through a couple of examples, the behaviour of these algorithms in terms of their performance is studied, especially focusing on the influence of the available choices when implementing them, such as, the codification of the chromosomes, the choice of a fitness function or the freedom in tuning the parameters.

The last chapter is entirely dedicated to merging genetic algorithms with quantum information. Indeed, two approaches are analysed concerning the merging of quantum computation and genetic algorithms. In the first one, I reviewed previous work [15], and I tried a different approach to obtain an improvement in the selection procedure of a classical genetic algorithm employing quantum searching algorithms. I numerically show that the proposal may resemble the evolutionary selection philosophy, making it monotonically increasing with the fitness criteria and not suppressing individuals with low fitness. In the future, it would be meaningful to compare the performance of each selection subroutine with different fitness functions, in order to better understand the conditions under which the approach shows and advantage with respect to previous one.

In the second part, we focused on the development of a fully quantum genetic algorithm. Each building block of the quantum genetic algorithm is explained by means of quantum gates. Finally, the developed algorithm is simulated in a classical computer and the results are positively compared against a classical version of the genetic algorithm. A further study of the properties of this algorithm must be carried out with other fitness functions and performance measurements in order to understand the limits of the approach.

Besides, the procedure could be implemented in different registers which could in fact be different connected quantum processors. That is, its structure is especially suitable for distributed quantum computing. If each register is in a different quantum processor, we would need to be able to perform SWAP operations between them. In the quantum genetic algorithm, SWAP is the only operation required between the registers, except for the pairwise sorting oracle. But the pairwise sorting oracle could be implemented in a master processor with twice the capacity. The importance of this stems from the difficulty to build big quantum processors, while building smaller ones is nowadays achievable.

I performed the calculations for the examples about genetic algorithms of the second chapter and the simulation of quantum algorithms of the third chapter with Python 3 programs, which are available in the repository of the University of the Basque Country (ADDI).

Future work would require a deeper analysis of the quantum genetic algorithm. Understanding its performance and its advantages could lead to more efficient optimization algorithms for some problems. Beyond the algorithm itself, the limitations of distributed quantum computing could also be tested employing this algorithm.

Bibliography

[1] M. Mitchell, *An introduction to genetic algorithms.* Cambridge, MA, USA: MIT Press, 1996.

[2] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information.* Cambridge: Cambridge University Press, 2000.

[3] N. Herbert, "FLASH-A superluminal communicator based upon a new kind of quantum measurement," *Foundations of Physics*, vol. 12, no. 12, pp. 1171–1179, 1982.

[4] W. K. Wootters and W. H. Zurek, "A single quantum cannot be cloned," *Nature*, vol. 299, pp. 802–803, 1982.

[5] V. Bužek and M. Hillery, "Quantum copying: Beyond the no-cloning theorem," *Physical Review A - Atomic, Molecular, and Optical Physics*, vol. 54, no. 3, pp. 1844–1852, 1996.

[6] A. Ferraro, M. Galbiati, and M. G. Paris, "Cloning of observables," *Journal of Physics A: Mathematical and General*, vol. 39, no. 14, p. L219–L228, 2006.

[7] U. Alvarez-Rodriguez, M. Sanz, L. Lamata, and E. Solano, "Biomimetic cloning of quantum observables," *Scientific Reports*, vol. 4, pp. 4–7, 2014.

[8] L. K. Grover, "A fast quantum mechanical algorithm for database search," pp. 1–8, 1996. [Online]. Available: arXiv:quant-ph/9605043v3

[9] M. Boyer, G. Brassard, P. Høyer, and A. Tapp, "Tight bounds on quantum searching," *Fortschritte der Physik*, vol. 46, no. 4-5, pp. 493–505, 1998.

[10] C. Dürr and P. Høyer, "A Quantum Algorithm for finding the minimum," pp. 1–2, 1996. [Online]. Available: arXiv:quant-ph/9607014

[11] G. S. Hornby, A. Globus, D. S. Linden, and J. D. Lohn, "Automated antenna design with evolutionary algorithms," *Collection of Technical Papers - Space 2006 Conference*, vol. 1, pp. 445–452, 2006.

[12] J. H. Holland, *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence.* Cambridge, MA, USA: MIT Press, 1992.

[13] K. A. De Jong, "Genetic Algorithms Are NOT Function Optimizers," in *Foundations of Genetic Algorithms*, ser. Foundations of Genetic Algorithms, 1993, pp. 5–17.

[14] R. S. Zebulum, M. A. C. Pacheco, and M. M. B. Vellasco, *Evolutionary electronics: Automatic design of electronic circuits and systems by genetic algorithms*, 2001.

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