QGA: A QUANTUM GENETIC ALGORITHM

Andrea Malossini, Enrico Blanzieri and Tommaso Calarco

December 2004

Technical Report # DIT-04-105

QGA: A Quantum Genetic Algorithm

ANDREA MALOSSINI
University of Trento

ENRICO BLANZIERI
University of Trento

TOMMASO CALARCO
Istituto Nazionale per la Fisica della Materia, BEC-INFM Trento
ECT* - European Centre for Theoretical studies in nuclear physics and related areas

December 9, 2004

Abstract

The complexity of the selection procedure of a genetic algorithm that requires reordering, if we restrict the class of the possible fitness functions to non–local or time–dependent fitness functions, is $\mathcal{O}(N \log N)$ where N is the size of the population. Quantum Genetic Algorithm (QGA) exploits the power of quantum computation in order to speed up genetic procedures. In QGA the classical fitness evaluation and selection procedures are replaced by a single quantum procedure. QGA outperforms a typical classical genetic algorithm. We show that the complexity of our QGA is $\mathcal{O}(1)$ in terms of number of oracle calls in the selection procedure. Such theoretical results are confirmed by the simulations of the algorithm.

1 Introduction

The possible interplay between quantum and genetic algorithms has been only partially explored. On the one hand, quantum algorithms exploit the laws of quantum mechanics in order to perform efficient computation. It has been shown that quantum computation can dramatically improve performance for solving problems like factoring [Sho94] or searching in an unstructured database [Gro97]. On the other hand, genetic algorithms [Gol89] can be described, basically, as search algorithms. They work on a set of elements, called *population*, that evolves, by means of crossover and mutation, towards a maximum of the fitness function. Since their proposal, genetic algorithms have proved to be efficient and flexible algorithms for solving a wide range of problems. The need to have fast implementations of genetic algorithms is testified by the great number of hardware implementations for them [AC01]. In this perspective, having a quantum version of a genetic algorithm seems to be a relevant topic in the future, when quantum computers will be available. Another motivation for an integration between the two paradigms comes from the fact that this could be a way of applying quantum computation to hard problems [DJS89] for which a quantum algorithm is not available yet.

The possibility to integrate the quantum and genetic algorithms has been recently proposed by [HK02]. The authors presented a parallel evolutionary algorithm with a quantum representation and a coarse–grained parallel scheme. By adopting a qubit chromosome representation, a quantum population is generated. Classical populations are generated by performing measurements on the quantum population; then the best elements are searched for in the classical population and used to update the quantum population. The process iterates until the requirements are

QGA	Quantum Genetic Algorithm
$\mathbb{C}$	Complex space
$\vert\cdot\rangle$	A Hilbert space vector
U	A unitary operator
U_F	Unitary operator for fitness evaluation
$\otimes$	Tensor product of Hilbert spaces
N	Number of elements of a population, or number of database entries
M	Number of genetic steps
n	Number of qubits
$H^{\otimes n}$	n–qubit Hadamard–Walsh gate
n_{h}	Number of Dürr–Høyer iterations
κ_{BBHT}	Constant appearing in the BBHT algorithm
$f(\cdot)$	Fitness function
$F(\cdot)$	Fitness function, in function of the index
F_i	Fitness function value
t	Number of marked solutions
$\theta(\cdot)$	Heaviside function
$\mathbb{E}(\cdot)$	Expectation value
$\sum_j$	$\sum_{j=0}^{N-1}$

Table 1: Notation used in this paper.

fulfilled. Han and Kim showed also a parallel version. However, the algorithm they developed is not a quantum algorithm, but a novel evolutionary algorithm for a classical computer based on the principles of quantum computation. An empirical analysis of the performance of their algorithm is provided but no time–complexity analysis is reported.

In this paper, we present a quantum genetic algorithm (QGA), a quantum algorithm that exploits the power of quantum computation in the fitness evaluation and selection procedures, and we show how to take advantage of quantum phenomena to efficiently speed up classical computation. We exploit the power of quantum computation not only to represent the population by means of qubits, but also to perform fitness evaluation and selection. The algorithm is based on the Dürr–Høyer quantum algorithm for finding the minimum in an unsorted table [DH96]. Our results rely on the observation that it is possible to stop the quantum procedure of the Dürr–Høyer algorithm and use the partial result for the selection. A theoretical description of QGA is provided as well as a detailed analysis of the algorithm complexity. In particular, we show that the complexity of the quantum selection procedure (which includes the quantum fitness evaluation) does not depend on the size of the population N. Moreover, we show that the convergence speed, in terms of genetic steps, of the quantum genetic algorithm is comparable to the convergence speed of a classical steady–state genetic algorithm with truncation selection. Finally we provide a simulation of the algorithm, which fully validates the theoretical results.

The remaining of the present section is devoted to introduce the concepts related to genetic and quantum computation that are necessary for presenting the algorithm. In Table 1 we present the notations used in the paper. Section 2 presents our QGA. Section 3 presents the analysis of the complexity whereas Section 4 is devoted to simulation of the algorithm and empirical validation of the theoretical results. Finally, we draw some conclusions in Section 5.

1.1 Introduction to Genetic Algorithms

Genetic algorithms are adaptive search algorithms based on the evolutionary ideas of natural selection and genetics. They are based on the principle first laid down by Charles Darwin of survival of the most fit. First pioneered by John Holland [Hol75], genetic algorithms have been widely studied, experimented and applied in many fields. A generic steady–state genetic algorithm is sketched in Fig. 1. The first step is the creation of a random population where each element is coded using a specific representation that encodes a set of features defined by the problem. Then a *fitness function* is used to evaluate each individual and the reproductive success varies with the fitness value. Two high–fitness elements are chosen for crossover and mutation. The procedure generates two new offsprings that replace two random elements of the population. The process continues until the population's total fitness gets over a specified threshold or the number of genetic steps reaches a predefined value.

In genetic algorithms the *fitness function* of the problem leads the population to converge toward a population that fits the solution requirements. For complex problems the definition of an exact fitness function that describes perfectly the nature of the problem is often not possible and we are forced to use approximate fitness functions. This implies that during the selection procedure we cannot discriminate between two individuals with almost the same fitness value and a more fruitful approach is to select a fraction of high–fitness individuals and use them for generating new offsprings. This selection procedure is called *truncation selection* [MSV93, MSV94]. In the *generational* approach a new population is generated at every genetic step which substitutes the old population. In the *incremental* (or *steady–state*) approach only two new offsprings are generated at every genetic step and inserted in the population. The later approach is needed when the fitness value of an element of a population depends on all the individuals of the population (*non–local* fitness function) or the fitness function is *time–dependent* and changes at every genetic step. We will see in Section 4 that when it is not possible to mantain an order between fitness values of individuals, the power of quantum computation in performing unstructured search will contribute to develop an efficient quantum genetic algorithm.

1.2 Introduction to Quantum Search Algorithms

The basic unit of information of quantum computation is the *qubit*. A qubit is a two–level quantum system and it can be represented by a unit vector of a two dimensional Hilbert space ($\alpha, \beta \in \mathbb{C}$):

$$|\,\psi\,\rangle = \alpha|\,0\,\rangle + \beta|\,1\,\rangle, \qquad |\alpha|^2 + |\beta|^2 = 1$$

where we denote with $|\,0\,\rangle$ and $|\,1\,\rangle$ the basis states adopting the *ket notation* for quantum systems. A two–level quantum system is described by a superposition of the basis states whereas a two–level classical system can be just in one of the basis states 0 or 1.

The evolution of a quantum system is described by special linear operators, *unitary operators*[1] U which operates on qubits.

$$U|\,\psi\,\rangle = U[\alpha|\,0\,\rangle + \beta|\,1\,\rangle] = \alpha\, U|\,0\,\rangle + \beta\, U|\,1\,\rangle.$$

An important consequence of the linearity of quantum operators is that the evolution of a two–level quantum system is the linear combination of the evolution of the basis states $|\,0\,\rangle$ and $|\,1\,\rangle$. This is known as *quantum parallelism*. On the contrary

[1]A linear operator is said to be *unitary* if $UU^\dagger = U^\dagger U = \mathbb{1}$, where $U^\dagger$ denotes the adjoint of the operator U.

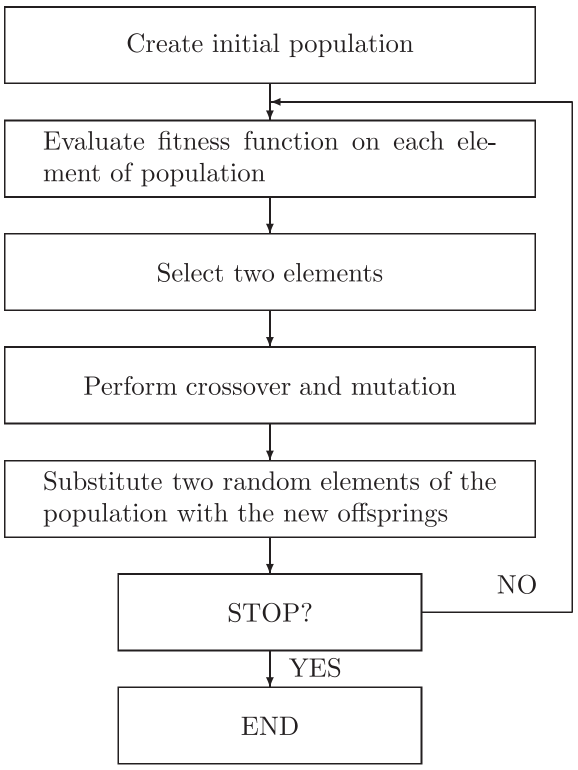
Create initial population
Evaluate fitness function on each element of population
Select two elements
Perform crossover and mutation
Substitute two random elements of the population with the new offsprings
STOP?
NO
YES
END

Figure 1: A typical steady–state genetic algorithm

in a two–level classical system we are forced to evolve the two possible states 0 and 1 separately. When we want to transfer information from the quantum system to a classical one, we have to perform *measurements* of the quantum state, whose result is probabilistic: we get the state $U|\,0\,\rangle$ with probability $|\alpha|^2$ and the state $U|\,1\,\rangle$ with probability $|\beta|^2$. So there is no way of knowing exactly both values, and we cannot either clone the unknown state $|\,\psi\,\rangle$ as stated by the *No cloning theorem* (see a formulation of it in [Per98]). This negates the possibility of knowing perfectly an unknown quantum state by doing a lot of replicas of it and measuring each of them.

Another important feature arising from the linearity of quantum mechanics is *entanglement*. The state of a composite classical system AB is completely determined by the state of its sub–systems. On the contrary, the state of a composite quantum system is the *tensor product* $\otimes$ of the states of the component systems; so a state of a composite system $|\,\psi\,\rangle_{\mathrm{AB}}$ could be like

$$|\,\mathrm{Bell}\,\rangle_{\mathrm{AB}} = \frac{1}{\sqrt{2}}[|\,0\,\rangle_{\mathrm{A}} \otimes |\,0\,\rangle_{\mathrm{B}} + |\,1\,\rangle_{\mathrm{A}} \otimes |\,1\,\rangle_{\mathrm{B}}],$$

which is not of the form $|\cdot\rangle_{\mathrm{A}} \otimes |\cdot\rangle_{\mathrm{B}}$. The *Bell state* is said to be *entangled*. The entanglement is a quantum resource that permits, for instance, quantum teleportation [BBC+93].

The two main quantum algorithms developed up to now are Quantum Fourier Transform (QFT) [Sho94], and Grover Search Algorithm [Gro97]. QFT can be used to solve problems like, discrete logarithm, order finding and factoring [NC00] and it lies out of the scope of this paper. Grover algorithm has been used in BBHT algorithm and Dürr–Høyer algorithm and we briefly review the three algorithms below.

1.2.1 Grover algorithm

The algorithm solves the problem of searching in an unstructured database. It has been shown that the Grover algorithm is $\mathcal{O}\left(\sqrt{N/t}\right)$ where N is the number of entries in the database and t is the number of possible solutions [Gro97]. Classical algorithms for solving this problem must, instead, look at each entry of the database until a solution is found, i. e. they are $\mathcal{O}\left(N/t\right)$. The basic idea of the Grover's algorithm is to amplify the coefficients of the superposition of all elements, which correspond to the solutions of the given problem, while reducing the others. This procedure is performed by applying a unitary operator $\mathcal{O}\left(\sqrt{N/t}\right)$ times. Then a measurement of the quantum state obtained will yield, with high probability, one of the possible solutions. The non–structuredness requirement is essential for achieving the speed–up stated above, otherwise classical binary tree search would solve the problem in $\mathcal{O}\left(\log N\right)$. It should be emphasized that a classical procedure always permits to collect all the solutions in the database (by seeking all the entries), on the contrary the probabilistic nature of quantum measurement allows to get one solution at random among the solutions of the database. However, by repeating the whole quantum procedure it is possible to obtain other solutions.

1.2.2 BBHT algorithm

When the number of solutions is known in advance, we can use Grover algorithm to look for one of them. Without previous knowledge of the number of solutions t marked by the oracle we cannot use Grover algorithm. This impossibility arises because in the amplitude amplification process we cannot compute the number of iterations to be performed in order to maximize the coefficients of the solution. However, when the number of solutions t is a priori unknown, it is still possible to use a remarkable quantum algorithm called BBHT[BBHT98] for finding a solution in a set of items $\{T_i\}_{i=0,\ldots,N-1}$ given an oracle that recognizes a solution.

Here we give a brief summary of the BBHT algorithm and report the main complexity result. We suppose, at first, that $1 \leq t \leq 3N/4$, where N is the total number of elements.

1. Initialize $m = 1$, set $\lambda = 6/5$ (any value between 1 and 4/3 would do) and create the state $|\,\Psi_0\,\rangle = H^{\otimes n}|\,0\,\rangle = \frac{1}{\sqrt{N}}\sum_j |\,j\,\rangle$.

2. Choose i uniformly at random among the nonnegative integers smaller than m.

3. Apply i iterations of Grover's algorithm starting from initial state $|\,\Psi_0\,\rangle$.

4. Measure the register: let o be the outcome.

5. If the selected element T_o is a solution then `exit`.

6. Otherwise, set m to $\min(\lambda m, \sqrt{N})$ and go back to step 2.

The case $t > 3N/4$ can be treated in constant time by classical sampling.

Theorem 1.1. *The BBHT algorithm finds a solution in an expected time of* $\mathcal{O}\left(\sqrt{N/t}\right)$.

Proof. See [BBHT98]. □

Remark 1.2. *As a step of the proof the authors showed that the number of oracle queries is bounded from above by* $4\sqrt{N/t} = \kappa_{\text{BBHT}}\sqrt{N/t}$ *when* $t \ll N$.

1.2.3 Dürr–Høyer algorithm

The Dürr–Høyer algorithm is a quantum algorithm for finding the minimum within an unsorted table of N items [DH96]. The core of the algorithm is a procedure which returns an index of an item smaller than the item determined by a particular threshold index by using the BBHT algorithm. This procedure is iterated until the minimum is reached. Dürr and Høyer showed that such an algorithm requires an expected number of $\mathcal{O}\left(\sqrt{N}\right)$ iterations.

1.2.4 Quantum evaluation of functions

Like a small set of classical gates (e.g. AND OR NOT) can be used to compute an arbitrary classical function, a similar universality result is true for quantum computation. A set of gates is said to be *universal for quantum computation* if any unitary operation may be approximated to arbitrary accuracy by a quantum circuit involving only those gates. It has been shown that using *Hadamard, phase, CNOT and* $\pi/8$ gates, any arbitrary unitary operation can be approximated to arbitrary accuracy [NC00, pag. 188].

Moreover, any classical circuit can be made reversible by introducing a special gate named *Toffoli gate*. The Toffoli gate has three input bits, a, b, and c; a and b are the first and the second "control bits", while c is the "target bit". The gate does not change the control bits and flips the target bit only if both control bits are set. The Toffoli gate can be used to implement NAND and FANOUT and it is reversible. A quantum version of Toffoli gate has been introduced (see e.g. [NC00, pag. 182]). Hence, given a classical reversible circuit that computes a function $f(x)$ we can imagine to convert such a circuit in a quantum one obtaining a *quantum evaluation* of $f(x)$.

2 A Quantum Genetic Algorithm

The basic structure of our quantum genetic algorithm is based on the classical structure of a typical steady–state genetic algorithm. In particular we have developed a quantum selection procedure that includes a quantum fitness evaluation unit. In Fig. 2 a comparison between the classical genetic algorithm and the quantum genetic algorithm is shown. Notice that no external quantum evaluation procedure is needed since quantum fitness recalculation is computed inside the quantum selection procedure. This procedure is based on the quantum algorithm for finding the minimum proposed by [DH96] who showed that it is possible to find the minimum of a list by using a variant of the Grover quantum search algorithm in $\mathcal{O}\left(\sqrt{N}\right)$. By reducing the number of iterations we show that we can select a sub–population of optimal elements in constant time and that the convergence speed, in terms of genetic steps, of such algorithm is comparable to convergence speed of a classical steady–state genetic algorithm with truncation selection. The main difference is that, in the quantum selection procedure, in each genetic step the choice of a optimal sub–population is performed in constant time whereas in a classical selection procedure an ordering algorithm is needed.

2.1 Quantum fitness evaluation unit

As explained in the introduction, given a classical reversible circuit that computes a fitness function $F(j) = F_j$, where $j \in \{0, \ldots, N-1\}$ are the elements of the population in binary representation, we can convert it to a quantum one obtaining a *quantum fitness evaluation*. Let us suppose to have a quantum black box, whose

CLASSICAL GENETIC ALGORITHM

Create population: Generate N strings representing possible solutions of the problem.
Repeat M times

- **Evaluate fitness**: Evaluate the fitness of every element of the population.
- **Select two elements**: Select a subpopulation using *truncation selection* (a fraction p of the best elements of the population) and then choose randomly two elements from it.
- **Crossover and mutation**: Perform crossover of the two elements by exchanging two random substrings. Then with probability P_{M} mutate each allele (i.e bit) of the strings.
- **Substitution**: Choose two random elements from the population and then replace them with the new offsprings.

QUANTUM GENETIC ALGORITHM

Create population: As above.
Repeat M times

- **Select two elements**: Use the *quantum selection procedure* (which performs the creation of a superposition of all elements of the population and the application of the quantum fitness evaluation unit) to choose one element. Run it again to choose another element.
- **Crossover and mutation**: As above.
- **Substitution**: As above.

Figure 2: Classical and Quantum genetic algorithm. Note that we need to run the selection procedure twice because the measurement process destroys the superposition of the elements.

internal construction is not of concern here, that is able to compute the fitness function of the elements of the population; if we use the quantum binary encodings[2] for the elements, the superposition of all elements of the population is denoted by

$$|\,\Psi\,\rangle = \sum_{j=0}^{N-1} |\,j\,\rangle,$$

and the action of the quantum black box results in

$$U_F|\,\Psi\,\rangle|\,0\,\rangle = \sum_{j=0}^{N-1} |\,j\,\rangle|\,F_j\,\rangle,$$

where U_F denotes the unitary operator for fitness evaluation. Hence, using U_F only once we compute all the fitness values $\{F_j \mid j = 0, \dots, N-1\}$ of the population, whereas the classical procedure requires N fitness evaluation. The process of measurement would destroy such superposition giving us only one fitness value. So at this stage we could not gain any useful information on the best elements of the population. The oracle of the quantum selection procedure includes this unit to "mark" all the elements of the population that fulfill the condition $F_j \geq F_y$ where y is a threshold index.

2.2 Quantum selection procedure

The quantum selection procedure is based on the algorithm of [DH96] for finding the minimum of a list of N items. The authors showed that for finding the (absolute) minimum, a number of iterations $\mathcal{O}\left(\sqrt{N}\right)$ is needed. Here, we are not interested in finding the minimum, but in selecting a sub–population of near–optimal elements of the whole population, namely elements with relatively high value of fitness. The algorithm works as follow:

```
QUANTUM SELECTION PROCEDURE

1. Choose randomly a index $y \in \{0, 1, \dots, N-1\}$ corresponding to the
   threshold $F_y$. Compute classically $F_y = F(y)$.
2. Perform $n_{\mathrm{h}}$ times:
   (a) Initialize memory to $|\,0\,\rangle|\,y\,\rangle$.
   (b) Perform the algorithm BBHT (the step 1 of BBHT transforms the
       state $|\,0\,\rangle|\,y\,\rangle$ into $\frac{1}{\sqrt{N}}\sum_j |\,j\,\rangle|\,y\,\rangle$), where the oracle (that includes
       a quantum fitness evaluation unit) inverts the amplitude of the
       elements that satisfy $F_j \geq F_y$.
   (c) Measure the first ket and get a new index $y'$. Compute classically
       $F_{y'} = F(y')$. If $F_{y'} > F_y$ then set the index $y$ to $y'$.
3. Return the index $y$.
```

Definition 2.1. *A* Dürr–Høyer iteration *is the sequence of operations defined in 2a, 2b, 2c of the Quantum Selection Algorithm. We denote with n_{h} the number of Dürr–Høyer iterations.*

[2]Given $j = b_0 * 2^0 + b_1 * 2^1 + \cdots + b_{n-1} * 2^{n-1}$ where $b_i \in \{0, 1\}$, then $|\,j\,\rangle = |\,b_0\,\rangle \otimes |\,b_1\,\rangle \otimes \cdots \otimes |\,b_{n-1}\,\rangle$

Remark 2.2. *When $n_{\mathrm{h}} = 1$, we obtain the* BBHT *algorithm. Dürr and Høyer analyzed the case $n_{\mathrm{h}} = \infty$.*

One might argue that a probabilistic algorithm could do about the same, by choosing $\mathcal{O}(\log R)$ elements, where R is a fraction of the entire population, evaluating the fitness function for the chosen elements (and only for them), and picking the best one. Such assumption is not correct since the convergence of the genetic algorithm is different for the two selection procedures. After n_{h} iterations we have a probability for choosing the best element of the population equal to R/N; instead in this classical probabilistic algorithm the probability would be only $\log R/N$ (exponentially smaller). The main difference is that in one case we choose among the best elements, in the other we choose in a completely random way.

3 Complexity of the algorithm

In this section we present a complexity analysis of the Quantum Genetic Algorithm, in order to compare it with a classical genetic algorithm. We do not consider the computational cost of crossover, mutation and substitution of the QGA because they are constant for each genetic step and classical for both algorithms, and concentrate our analysis on the quantum selection procedure, whose time–complexity in terms of oracle calls will be deeply investigated. The time required for a single oracle call will depend on the technology used for implementing the oracle.

Let us consider the complexity of the quantum selection procedure step by step. Steps (1) and (3) of the Quantum Selection Procedure do not enter in the complexity calculation since they are performed only once and in constant time. Step (2a) inizializes the quantum memory and it is performed n_{h} times. Step (2c) performs the measurement process and it requires n_{h} classical computations of the fitness function. Step (2b), in terms of number of n-qubit operators, is the most onerous and, from the point of view of the complexity, it requires a deeper analysis. We will analyze this step in terms of number of *oracle calls*. The oracle includes the quantum fitness evaluation unit and inverts the amplitude of the elements with fitness greater or equal to a given threshold F_y. We will consider an oracle call as the *time step unit* for our analysis of step (2b) without taking into account steps (1), (3), (2a) and (2c), because their cost depends linearly on n_{h} and does not depend on the number of qubits $n = \log N$.

We are interested in the expected number of oracle calls in the quantum selection procedure; it is known that the BBHT algorithm requires $\mathcal{O}\left(\sqrt{N/t}\right)$ oracle calls where t is the number of marked elements (see Theorem 1.1). Dürr and Høyer found that the expected number of oracle calls of their algorithm in order to find the minimum is $22.5\sqrt{N}$. In our algorithm the number of Dürr–Høyer iterations is a parameter and we need to characterize its relation with the expected number of oracle calls. We will show in this section (Theorem 3.4) that the expected number of oracle calls is bounded from above by $\kappa \cdot 2(2^{n_{\mathrm{h}}} - 1)$ where κ is a constant and n_{h} is the number of Dürr–Høyer iterations. This is our main result because it states that the expected number of oracle calls does not depend on the dimension of the population N. In order to show this result we will need a bound on the expected number of oracle calls (Theorem 3.3). Moreover we will show that n_{h} is directly related to the selection pressure (Theorem 3.7).

In order to characterize the expected number of oracle calls of step (2b) of the *Quantum selection procedure*, we need to prove a Lemma. We consider a list of N elements and a fitness function f that maps each element onto a real positive value. We define as the *rank* of an element the position $s \in \{1, N\}$ of the element in the list sorted in descending order of the fitness function values.

Lemma 3.1. *The probability of choosing an element of rank s as threshold before the m-th Dürr–Høyer iteration* $\Pr(s, m)$, *is*[3]

$$\Pr(s,m) = \begin{cases} \dfrac{1}{N} \quad \text{if } m = 1 \\ \displaystyle\sum_{\substack{j_m=1,\\ j_{m-1}=1,\\ \dots,\\ j_2=1}}^{N} \frac{\theta(j_m - s)\theta(j_{m-1} - j_m)\cdots\theta(j_2 - j_3)}{j_m \cdot j_{m-1} \cdots j_3 \cdot j_2 \cdot N} \quad \text{otherwise} \end{cases} \tag{1}$$

Proof. We denote with $\Pr(s, l)$ the probability that we choose an element of rank s before the $l-$th Dürr–Høyer iteration; with $\Pr(s \mid j, l)$ the conditional probability that we choose an element of rank s before the $l-$th Dürr–Høyer iteration, after an element of rank j has been chosen in the previous iteration. We use the *total probability* equation

$$\Pr(s,l) = \sum_{j=1}^{N} \Pr(s \mid j, l) \cdot \Pr(j, l-1),$$

which holds because the set of possible events "choosing an element of rank j" is a partition of the set of events. During each Dürr–Høyer iteration we have that $\Pr(s|j, l) = \frac{1}{j}$ if $s \leq j$ or zero otherwise as ensured by step 3 of the algorithm:

$$\Pr(s|j,l) = \frac{\theta(j-s)}{j}.$$

The first index is chosen uniformly at random from all the elements, so $\Pr(s, 1) = \frac{1}{N}$, where $N = 2^n$. Using the *total probability* equation recursively we finally obtain that

$$\begin{aligned} \Pr(s,2) &= \sum_{j_2=1}^{N} \Pr(s \mid j_2, 2) \cdot \Pr(j_2, 1) \\ &= \sum_{j_2=1}^{N} \theta(j_2 - s) \frac{1}{j_2 \cdot N} \\ \Pr(s,3) &= \sum_{j_3=1}^{N} \Pr(s \mid j_3, 3) \cdot \Pr(j_3, 2) \\ &= \sum_{j_3=1}^{N} \sum_{j_2=1}^{N} \frac{\theta(j_3 - s)\theta(j_2 - j_3)}{j_3 \cdot j_2 \cdot N} \\ &\vdots \\ \Pr(s,m) &= \sum_{\substack{j_m=1,\\ j_{m-1}=1,\\ \dots,\\ j_2=1}}^{N} \frac{\theta(j_m - s)\theta(j_{m-1} - j_m)\cdots\theta(j_3 - j_4)\theta(j_2 - j_3)}{j_m \cdot j_{m-1} \cdots j_3 \cdot j_2 \cdot N} \end{aligned}$$

□

Definition 3.2. *Let $\mathcal{N}_m$ be the random variable* number of oracle calls during the m-th Dürr–Høyer iteration*. Moreover, let $\mathcal{N}$ be the random variable* total number of oracle calls in the quantum selection procedure*.*

[3] $\theta(x) = \begin{cases} 1 & x \geq 0 \\ 0 & \text{otherwise} \end{cases}$.

The following theorem uses the previous Lemma in order to bound the expected number of oracle calls.

Theorem 3.3. *The expectation of the total number of oracle calls in the quantum selection procedure is*

$$\mathbb{E}[\mathcal{N}] \leq \frac{\kappa}{\sqrt{N}} \sum_{s=1}^{N} \frac{1}{\sqrt{s}} \cdot \left[1 + \sum_{m=2}^{n_\mathrm{h}} \sum_{\substack{j_m=1,\\ j_{m-1}=1,\\ \ldots,\\ j_2=1}}^{N} \frac{\theta(j_m - s)\,\theta(j_{m-1} - j_m) \cdots \theta(j_2 - j_3)}{j_m \cdot j_{m-1} \cdots j_3 \cdot j_2} \right] \tag{2}$$

where κ is a constant.

Proof. From the very definition of expectation and Theorem 1.1, the expected number of oracle calls during the $m - th$ Dürr–Høyer iteration is

$$\mathbb{E}[\mathcal{N}_m] \leq \sum_{s=1}^{N} \kappa \sqrt{\frac{N}{s}} \cdot \Pr(s, m)$$

using Lemma 3.1 (Eq. 1) we show that

$$\mathbb{E}[\mathcal{N}_m] \leq \begin{cases} \dfrac{\kappa}{\sqrt{N}} \displaystyle\sum_{s=1}^{N} \dfrac{1}{\sqrt{s}} & \text{if } m = 1 \\ \dfrac{\kappa}{\sqrt{N}} \displaystyle\sum_{\substack{s=1,\\ j_m=1,\\ j_{m-1}=1,\\ \ldots,\\ j_2=1}}^{N} \dfrac{\theta(j_m - s)}{\sqrt{s}} \left[\displaystyle\prod_{l=3}^{m} \dfrac{\theta(j_{l-1} - j_l)}{j_l} \cdot \dfrac{1}{j_2} \right] \end{cases} \tag{3}$$

From the definition of $\mathcal{N}$ it is clear that

$$\mathcal{N} = \sum_{m=1}^{n_\mathrm{h}} \mathcal{N}_m,$$

whence we obtain

$$\mathbb{E}[\mathcal{N}] \leq \sum_{m=1}^{n_\mathrm{h}} \mathbb{E}[\mathcal{N}_m].$$

□

The bound of Theorem 3.3 depends on the number of elements of the population. We now want to calculate another upper bound of the expectation of $\mathcal{N}$. In particular, this upper bound it is independent of the cardinality of the population, as stated by the following theorem.

Theorem 3.4. *The expected number of oracle calls in the quantum selection procedure is bounded by*

$$\mathbb{E}[\mathcal{N}] < \kappa \cdot 2\left(2^{n_\mathrm{h}} - 1\right), \tag{4}$$

where κ is a decreasing function of n_h.

Proof. First we show that for all $m \in \{1, N\}$

$$\mathbb{E}[\mathcal{N}_m] < \kappa \cdot 2^m. \tag{5}$$

From calculus we have that

$$\sum_{s=1}^{N} \frac{1}{\sqrt{s}} < 1 + \int_{1}^{N} \frac{1}{\sqrt{s}}\,\mathrm{d}s = 2\sqrt{N} - 1 < 2\sqrt{N}.$$

Taking $\kappa = 1$ for notation semplicity, for $m = 1$, from Eq. 3:

$$\frac{1}{\sqrt{N}} \sum_{s=1}^{N} \frac{1}{\sqrt{s}} < \frac{1}{\sqrt{N}}(2\sqrt{N} - 1) < 2 \quad ;$$

For $m > 1$:

$$\begin{aligned}
&\sum_{s=1}^{N} \sqrt{\frac{N}{s}} \cdot \sum_{\substack{j_m=1,\\ j_{m-1}=1,\\ \ldots,\\ j_2=1}}^{N} \frac{\theta(j_m - s)\,\theta(j_{m-1} - j_m)\cdots\theta(j_2 - j_3)}{j_m \cdot j_{m-1} \cdots j_3 \cdot j_2 \cdot N} \\
&= \frac{1}{\sqrt{N}} \sum_{j_2=1}^{N} \sum_{j_3=1}^{j_2} \cdots \sum_{j_m=1}^{j_{m-1}} \sum_{s=1}^{j_m} \frac{1}{\sqrt{s}} \cdot \frac{1}{j_m \cdot j_{m-1} \cdots j_3 \cdot j_2} \\
&< \frac{1}{\sqrt{N}} \sum_{j_2=1}^{N} \sum_{j_3=1}^{j_2} \cdots \sum_{j_m=1}^{j_{m-1}} 2\sqrt{j_m} \cdot \frac{1}{j_m \cdot j_{m-1} \cdots j_3 \cdot j_2} \\
&= \frac{2}{\sqrt{N}} \sum_{j_2=1}^{N} \sum_{j_3=1}^{j_2} \cdots \sum_{j_m=1}^{j_{m-1}} \frac{1}{\sqrt{j_m}} \cdot \frac{1}{j_{m-1} \cdots j_3 \cdot j_2} \\
&< \frac{2^2}{\sqrt{N}} \sum_{j_2=1}^{N} \sum_{j_3=1}^{j_2} \cdots \sum_{j_{m-1}=1}^{j_{m-2}} \sqrt{j_{m-1}} \cdot \frac{1}{j_{m-1} \cdots j_3 \cdot j_2} \\
&\vdots \\
&< \frac{2^{m-1}}{\sqrt{N}} \sum_{j_2=1}^{N} \frac{1}{\sqrt{j_2}} < \frac{2^{m-1}}{\sqrt{N}} \cdot 2\sqrt{N} = 2^m.
\end{aligned}$$

Now using Theorem 3.3 and the above results,

$$\mathbb{E}\left[\mathcal{N}\right] < \kappa \sum_{m=1}^{n_\mathrm{h}} 2^{\,m} = \kappa \cdot 2\,(2^{\,n_\mathrm{h}} - 1)$$

□

Remark 3.5. *It is important to emphasize that the bound depends on* n_h *only and does not depend on the dimension of the population* N*.*

We have seen that the number of Dürr–Høyer iterations n_h determines the upper bound to the number of oracle calls during the quantum selection; it is an important parameter of our algorithm and we want to understand deeply its meaning.

Definition 3.6. *We denote with* $\mathcal{T}_m$ *the random variable* number of marked elements after the m-th Dürr–Høyer iteration.

Theorem 3.7. *Let* $N = 2^{\,n}$*, the expected number of marked elements after* m *Dürr–Høyer iterations is*

$$\mathbb{E}\left[\mathcal{T}_m\right] = 1 + (2^{\,n} - 1) \cdot 2^{\,-m}. \tag{6}$$

Proof. For $m = 1$, $\mathbb{E}[\mathcal{T}_1] = \sum_{s=1}^{N} s \cdot \Pr(s,1) = (N+1)/2$. For $m > 1$, we change the order of summation and obtain that

$$
\begin{aligned}
\mathbb{E}[\mathcal{T}_m] &= \sum_{s=1}^{N} s \cdot \Pr(s,m) \\
&= \sum_{s=1}^{N} s \cdot \sum_{\substack{j_m=1,\\ j_{m-1}=1,\\ \dots,\\ j_2=1}}^{N} \frac{\theta(j_m - s)\,\theta(j_{m-1} - j_m)\cdots\theta(j_2 - j_3)}{j_m \cdot j_{m-1} \cdots j_3 \cdot j_2 \cdot N} \\
&= \sum_{j_2=1}^{N} \sum_{j_3=1}^{j_2} \cdots \sum_{j_m=1}^{j_m-1} \sum_{s=1}^{j_m} s \cdot \frac{1}{j_m \cdot j_{m-1} \cdots j_3 \cdot j_2 \cdot N} \\
&= \sum_{j_2=1}^{N} \sum_{j_3=1}^{j_2} \cdots \sum_{j_m=1}^{j_{m-1}} \frac{j_m(j_m+1)}{2} \cdot \frac{1}{j_m \cdot j_{m-1} \cdots j_3 \cdot j_2 \cdot N} \\
&= \frac{1}{2 \cdot N} \sum_{j_2=1}^{N} \sum_{j_3=1}^{j_2} \cdots \sum_{j_m=1}^{j_{m-1}} \frac{j_m + 1}{j_{m-1} \cdots j_3 \cdot j_2} \\
&= \frac{1}{2 \cdot N} \sum_{j_2=1}^{N} \sum_{j_3=1}^{j_2} \cdots \sum_{j_{m-1}=1}^{j_{m-2}} \cdot \left(\frac{j_{m-1}(j_{m-1}+1)}{2} + j_{m-1} \right) \cdot \frac{1}{j_{m-1} \cdots j_3 \cdot j_2} \\
&= \frac{1}{4 \cdot N} \sum_{j_2=1}^{N} \sum_{j_3=1}^{j_2} \sum_{j_{m-1}=1}^{j_{m-2}} \frac{j_{m-1} + 3}{j_{m-2} \cdots j_3 \cdot j_2} \\
&\;\vdots \\
&= \frac{1}{2^{m-1} \cdot N} \sum_{j_2=1}^{N} (j_2 + 2^{m-1} - 1) \\
&= \frac{1}{2^{m-1} \cdot N} \left(\frac{N(N+1)}{2} + N \cdot (2^{m-1} - 1) \right) \\
&= \frac{N + 2^m - 1}{2^m}.
\end{aligned}
$$

□

With $m = n_\mathrm{h}$ Theorem 3.7 shows clearly how n_h determines the expected number of marked elements, and thus the selection pressure. The effect of Dürr–Høyer iterations is shown in Fig. 3. The cardinality of the selected sub–population decreases for increasing number of n_h.

4 Simulation

In this section we present the results of a simulation of the QGA in order to show the validity of Theorem 3.3 and Theorem 3.7 which bound the expected number of oracle calls and characterize the selection pressure respectively. We used a particular fitness function in order to compare the convergence speed of the total fitness of the QGA with respect to a classical genetic algorithm with truncation selection.

Simulations of the classical genetic algorithm and of the quantum genetic algorithm were performed using the symbolic language Mathematica™. The quantum fitness evaluation unit was simulated as a black box without modelling the quantum circuits. The maximum number of qubits used is $n = 8$, because beyond that

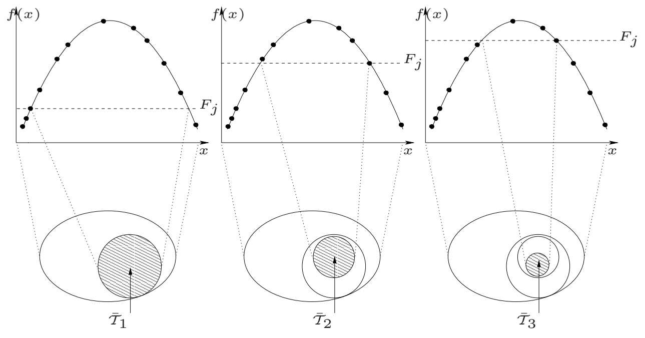

Figure 3: Change in the cardinality of the sub–population when n_h is changed from 1 to 3.

value too many computational resources were needed, since the resources needed to simulate a quantum computer on a classical one increase exponentially with n.

4.1 Fitness function

The fitness value of each element of the population reflects the quality of the characteristics that it encodes. As mentioned in Section 1 usually we have to model a simplified version of the problem due to the intrinsic complexity of a full treatment approach. In general, each element of a population could depend on all the other elements, and a small change in a single element could deeply influence the behavior of the others. This justifies the need for a non–local or time–dependent fitness function which has to be recalculated at each genetic step. This implies that no structure can be mantained during the computation to speed up the fitness evaluation procedure. Moreover it is quite common to have a noisy environment in which the problem is being studied, which means that the fitness function can change at every genetic step. We refer to the class of fitness functions which can vary at every genetic step as *non–local fitness functions.*

We have simulated a non–local fitness function by adding to the fitness value of an element a random quantity ϵ obtained from a Gaussian distribution of mean value 0 and variance $\sigma_\epsilon = 10 \cdot \mu$, where μ is the mutation probability.[4] The multi–peak non–local fitness function used in the simulations is

$$f(x) = \sin(\pi x) \cdot (9x \mod 1) + \epsilon_{\mathrm{Gaussian}(0,\sigma_\epsilon)}. \tag{7}$$

This function is plotted in Fig. 4.

4.2 Expected number of oracle calls

Eq. 2 gives a bound on the expected number of oracle calls in the quantum selection procedure. We recall that we need two elements of the population to cross over, so we have to run the quantum selection algorithm twice (or more if the elements concide) to obtain two different elements because the measurement process destroys the superposition. We can argue that for large population it suffices to run it only twice. To verify Eq. 2 we considered different population cardinalities $N = 2^n$, with $n = 2, 3, 4, 5, 6$. We have generated 100 random populations for each population cardinality and for $n_\mathrm{h} = 1, 2, 3, 4$, and we have run the quantum genetic algorithm

[4]If the value of σ_ϵ is too small, the mutation procedure masks the noisy effect of the noisy fitness function.

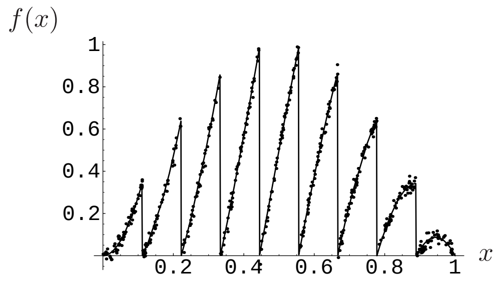

Figure 4: Main fitness function used in the simulation. The added Gaussian noise is also shown.

Table 2: Mean number of oracle calls and its standard deviation in the quantum selection procedure as results of the simulations.

n	$n_\mathrm{h} = 1$	$n_\mathrm{h} = 2$	$n_\mathrm{h} = 3$	$n_\mathrm{h} = 4$
2	6.3 ± 1.3	10.9 ± 1.6	-	-
3	6.4 ± 0.9	10.6 ± 1.3	13.4 ± 1.2	-
4	6.8 ± 0.8	10.8 ± 0.8	14.4 ± 0.8	18.2 ± 1.0
5	6.6 ± 0.4	11.2 ± 0.4	15.6 ± 0.6	21.0 ± 0.8
6	6.8 ± 0.3	11.7 ± 0.3	17.6 ± 0.4	25.1 ± 0.6

to select two offsprings. Results are shown in Table 2.[5] In order to verify the bound we need an estimate of the constant κ appearing in Eq. 2. Unfortunately an estimate is known only for $n_\mathrm{h} = 1$ and $t \ll N$ (BBHT algorithm). Our strategy was to fit the bound against the data and compare the values of the parameters. Then we ran a regression on the experimental points using Eq. 2 and estimated κ, as shown in Table 3. Finally, Fig. 5 shows the experimental plots (and error bars) and the regression function for different numbers of Dürr–Høyer iterations.

When $n_\mathrm{h} = 1$, our quantum selection procedure concides with BBHT (Remark 2.2), so it is interesting to compare the empirical value with the theoretical bound. From Remark 1.2, $\kappa_\mathrm{BBHT} \approx 4$. In Table 3 we obtain $\kappa_\mathrm{reg} = 3.79 \pm 0.08$ for $n_\mathrm{h} = 1$. But since in the selection procedure we need two different elements to crossover, we expect to use the quantum selection procedure on average at least twice. Hence $\kappa \leq 3.79/2 = 1.895 < 4 = \kappa_\mathrm{BBHT}$.

4.3 Performance comparison

Here we show that the convergence speed, in terms of genetic steps, of the quantum genetic algorithm is comparable to a classical truncation selection algorithm

[5]Some combinations of n and n_h are useless because the quantum genetic algorithm selects almost always the element with maximum fitness, being it impossible to cross over two different elements.

Table 3: Regression coefficients of the Eq. 2 data in Table 2.

n_h	Coefficient κ_reg	Coefficient of determination R^2
1	3.79 ± 0.08	0.9984
2	2.52 ± 0.07	0.9965
3	2.00 ± 0.03	0.9989
4	1.76 ± 0.02	0.9997

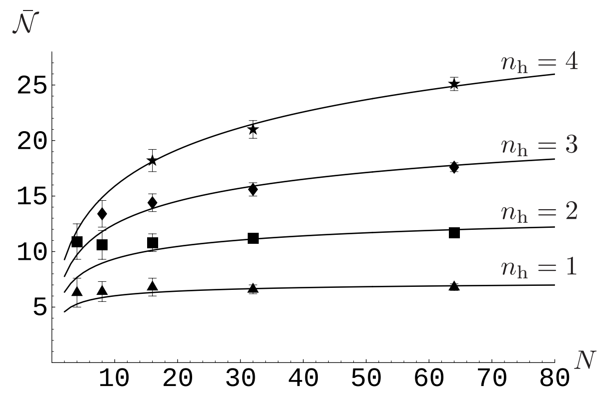

Figure 5: Mean number of oracle calls for different values of number of elements of the population N and with number of Dürr–Hoyer iterations $n_h = 1, 2, 3, 4$. Experimental data and fitted curves based on Theorem 3.4.

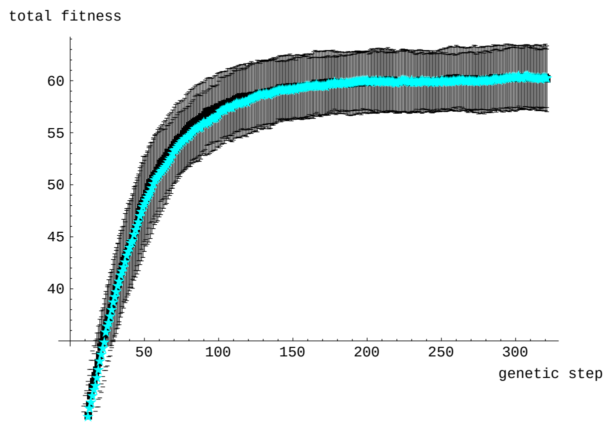

Figure 6: Total fitness (mean and variance) of Quantum genetic algorithm (brighter line) and classical genetic algorithm with truncation selection as function of the number of genetic steps. Each genetic step requires $\mathcal{O}(N \log N)$ in the classical selection procedure and $\mathcal{O}(1)$ in the quantum selection procedure.

where two elements of the fraction of the population are used to generate the new offsprings. This means that the total fitness function (the sum of all fitness values of the elements of the population) versus genetic steps should be equal within the statistical errors. The real power of the QGA is exploited at each genetic step where the computational complexity of the fitness selection procedure is $\mathcal{O}(1)$.

The number of genetic steps performed during the simulation is a multiple of $N/2$. After $N/2$ genetic steps we expect on average a complete change of the population (namely a new generation). Hence after M genetic steps we expect a number of generations $I \approx 2M/N$. The simulation has been performed using the same fitness function of Eq. 7 and with $I = 10$. The results of a simulation with a population of cardinality $64 = 2^6$ and $n_h = 3$ (i.e. about a fraction of 1/8 of the population at each genetic step) are shown in Fig. 6 and they confirm the analysis made.[6]

Finally, the regressions for the mean number of marked solution as a function of n_h and for different values of n are shown in Table 4; the corrisponding plot is

[6]We have done other simulations by changing the number of qubits and n_h; we obtain that the two curves are the same within the errors. See [Mal02].

Table 4: Regression of simulation data.

n	$\mathbb{E}[\mathcal{T}_{n_\mathrm{h}}]$	R^2
2	$(1.01 \pm 0.02) + (2.81 \pm 0.12) \cdot 2^{-n_\mathrm{h}}$	0.9816
3	$(0.98 \pm 0.03) + (7.76 \pm 0.19) \cdot 2^{-n_\mathrm{h}}$	0.9940
4	$(0.95 \pm 0.03) + (16.16 \pm 0.17) \cdot 2^{-n_\mathrm{h}}$	0.9989
5	$(1.06 \pm 0.11) + (30.53 \pm 0.69) \cdot 2^{-n_\mathrm{h}}$	0.9944
6	$(1.19 \pm 0.13) + (64.46 \pm 0.78) \cdot 2^{-n_\mathrm{h}}$	0.9986
7	$(0.96 \pm 0.52) + (125.81 \pm 2.53) \cdot 2^{-n_\mathrm{h}}$	0.9976

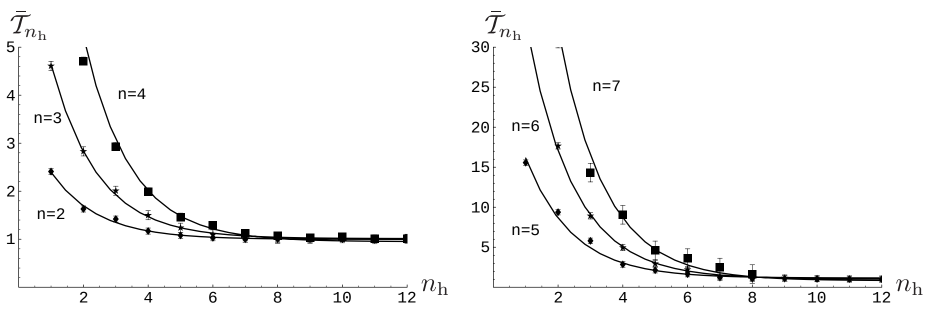

Figure 7: Mean number of marked elements for different values of number of Dürr–Høyer iteration. Experimental data and regression fitted curves based on Theorem 3.7.

shown in Fig. 7.

5 Conclusions

When the first quantum computers will start becoming available for applications, the need for quantum algorithms exploiting the power of such hardware will be pressing and the existing quantum algorithms will be subject to test. The number of quantum algorithms that fully exploit the power of quantum computation in order to gain significant speed-up is rather limited. Hence, a general approach for applying quantum computation to a wide range of problems is needed.

Our efforts in such direction have yielded to a quantum genetic algorithm that outperforms its classical analogue in terms of number of oracle calls. However, as explained above, starting from the complexity of Grover algorithm we know that we can speed up the process only if no structure is defined on the problem (hence the name "unstructered database search" used to refer to Grover quantum search algorithm).[7] Such requirement implies that, in order to achieve a quantum speed–up, we must restrict the problem class to time-dependent or non-local fitness functions, where the structure created by the evaluation of population elements is "broken" at every genetic step. In other words, *in order to gain a significant advantage over a classical approach using a quantum algorithm based on Grover search algorithm, we have to consider problems where the fitness function is non-local or time-dependent.*

Provided that, our QGA outperforms the classical one in terms of oracle calls. In fact, whereas the classical selection procedure requires $\mathcal{O}(N \log N)$ for reordering of the elements for non–local fitness functions, we have shown that the quantum selection procedure requires only $\mathcal{O}(1)$ quantum oracle calls. Our results do not

[7] In case of local a time–independent fitness function, a classical algorithm can order the initial results of fitness computation in $\mathcal{O}(N \log N)$ and maintain the order in $\mathcal{O}(\log N)$, exploiting such informations to speed up the computation.

contradict the well-known fact that in the black-box model the quantum speedup can be at most polynomial in the number of qubits. In fact, our algorithm does not search for a single marked element but for a fraction of marked elements with high fitness. The quantum fitness evaluation unit has to be implemented inside the quantum selection procedure which is performed twice and it computes the fitness in parallel on a superposition of elements at every genetic step. On the contrary a classical fitness evaluation has to be performed N times at every genetic step. We have to note that the quantum selection procedure selects the best elements of the population (the selection pressure depends on a parameter of the quantum genetic algorithm, n_h), and from them two elements are randomly chosen for the mating pool.

Truncation selection is one of the selection procedures used in classical genetic algorithms. It computes the fitness values of all the elements of the population, it orders them accordingly and it picks randomly two or more elements among a fraction of the best ones. The convergence speed of our algorithm, in terms of genetic steps, is comparable to a classical genetic algorithm with truncation selection, and the real power of quantum computation is exploited at every genetic step where the fitness evaluation and selection procedure are performed in $\mathcal{O}(1)$. Moreover the selection pressure of the algorithm can be controlled by a parameter of the QGA, n_h.

QGA is a quantum algorithm that combines the principles of genetic computation with the principles of quantum search. The result is that running on a quantum machine QGA will provide a sensible speed–up from $\mathcal{O}(N \log N)$ to $\mathcal{O}(1)$ on each genetic step where N is the dimension of the population. This result permits to use bigger populations as the number of qubits ($\log N$) used for the encoding will hopefully grow thanks to technology. The advantage will be far more useful for non-local time-dependent fitness function. When and how a quantum machine will be available is an open question. However, our proposal will permit to apply the advantages of quantum computation to a broader set of problems, provided a quantum fitness evaluation unit is designed.

References

[AC01] C. Aporntewan and P. Chongstitvatana. A hardware implementation of the compact genetic algorithm. In *Proceedings of the 2001 Congress on Evolutionary Computation CEC2001*, 2001.

[BBC+93] C. H. Bennett, G. Brassard, C. Crepeau, R. Jozsa, A. Peres, and W. Wooters. Telepoting an unknown quantum state via dual classical and epr channels. *Phys. Rev. Lett.*, 70:1895–1899, 1993.

[BBHT98] M. Boyer, G. Brassard, P. Høyer, and A. Tapp. Tight bounds on quantum searching. *Fortschr. Phys.*, 4(5):493–505, 1998.

[DH96] C. Dürr and P. Høyer. A quantum algorithm for finding the minimum. 1996.

[DJS89] K. A. De Jong and W. M. Spears. Using genetic algorithms to solve np-complete problems. In *In Proceedings of the Third International Conference on Genetic Algorithms*, pages 124–132, 1989.

[Gol89] D. E. Goldberg. *Genetic algorithms in search, optimization, and machine learning.* Addison-Wesley, 1989.

[Gro97] L. K. Grover. Quantum mechanics helps in searching for a needle in a haystack. *Phy. Rev. Lett.*, 72(2):325–328, 1997.

[HK02] K. H. Han and J. H. Kim. Quantum-inspired evolutionary algorithm for a class of combinatorial optimization. *IEEE Trans. on Evol. Comp.*, 6(6):580–593, 2002.

[Hol75] J. H. Holland. *Adaptation in natural and artificial systems.* The University of Michigan Press, 1975.

[Mal02] Andrea Malossini. Un algoritmo genetico di ricerca quantistica. Master's thesis, University of Padova (Italy), July 2002.

[MSV93] H. Mühlenbein and D. Schlierkamp-Voosen. Predictive models for the breeder genetic algorithm. *Evol. Comput.*, 1:25–49, 1993.

[MSV94] H. Mühlenbein and D. Schlierkamp-Voosen. The science of breeding and its application to the breeder genetic algorithm. *Evol. Comput.*, 1:335–360, 1994.

[NC00] M. A. Nielsen and I. L. Chuang. *Quantum computation and quantum information.* Cambridge Univ. Press, Cambridge, 2000.

[Per98] A. Peres. *Quantum theory: concepts and methods.* Kluwer Academic Publishers, Dordrecht, 1998.

[Sho94] P. W. Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings, 35th Annual Symposium on Foundations of Computer Science*, Los Alamitos, CA, 1994. IEEE Press.