Efficiency and performance analysis between Genetic Algorithm and Parallel Quantum Genetic Algorithm through the density classification and the Knapsack problems

Zakaria Laboudi[1], Salim Chikhi[2]
SCAL group, MISC Laboratory
Mentouri University
Route de Aïn El Bey 25017 Constantine
Algeria
[1]laboudizak@yahoo.fr, [2]slchikhi@yahoo.com

ABSTRACT: *Evolving solutions rather than computing them certainly represents a promising programming approach. Evolutionary computation has already been known in computer science since more than 4 decades. More recently, another alternative of evolutionary algorithms was invented: quantum genetic algorithms (QGA). In this paper, we outline the approach of QGA by giving a comparison with conventional genetic algorithm (CGA). We have executed our algorithm for 100 generations. Each CA in the population was iterated over M = 150 iterations. We have compared our best fitness obtained with the best that exists in the literature. Our parallel algorithm was run on a local network of 10 sites: a master site and nine slaves. The results have shown that QGA can be a very promising tool for exploring search spaces with a high performance and efficiency.*

Keywords: Genetic algorithm, Knapsack problem, Quantum genetic algorithm, Quantum computing, Cellular automata

Received: 8 June 2010, Revised 18 July 2010, Accepted 27 July 2010

1. Introduction

Genetic algorithms (GA) are a representative example of a set of methods known as evolutionary algorithms. This approach started in 1970 by John Holland, and knew for a decade strong growth. A GA is an iterative algorithm based on the notion of generation, but it is also inherently highly parallel in that it simulates the evolution of a range of solutions.

Quantum computation is a newly emerging interdisciplinary science of information science and quantum science. The first quantum algorithm was proposed by Shor in 1994 [14], for number factorization. Grover [6] also proposed a quantum algorithm for random search in databases, the complexity of its algorithm was reduced to be of the order of O(N ^ ½). More recently, quantum computation has attracted a wide attention, and it becomes a very interest research field.

Quantum genetic algorithm (QGA) is a combination of GA and quantum computing. There were some efforts to use QGA for exploring search spaces; we quote for example [4] where a QGA was used to solve the knapsack problem and [15] who proposed a parallel version of QGA. In [1], a QGA was also used to solve the binary decision diagram ordering problem. More recently, QGA where also used to evolve cellular automata rules (CA) [16] to solve the density classification problem.

In this work, we propose to make a comparison between GA and QGA to extract some computational abilities of QGA to perform processing in an effective and an efficient manner. Because QGA are highly parallel, we will show how can run them in a parallel environment. We have used two problems: the knapsack problem and the density classification problem. The first one was used to measure the effectiveness of QGA while the second one was used to measure their performance.

This paper is organized as follows. Section 2 describes some conventional GA principles. A description of the basic concept of quantum computing and QGA principles is presented in section 3. In section 4 we will make a comparison between GA and QGA by considering the 0/1 knapsack problem. Section 5 shows how can QGA be parallelized to solve the density classification problem by evolving cellular automata rules. In section 6 we summarize and analyze the experimental results. We finish our paper by concluding remarks follow and some perspectives in section 7.

2. Conventional genetic algorithm (CGA)

A Genetic algorithm is an exploration algorithm based on genetic evolution and natural selection. It manipulates a population of individuals called chromosomes. At each time step a new generation is constructed by applying genetic operators between some selected chromosomes. The structure of a CGA is illustrated in figure 1:

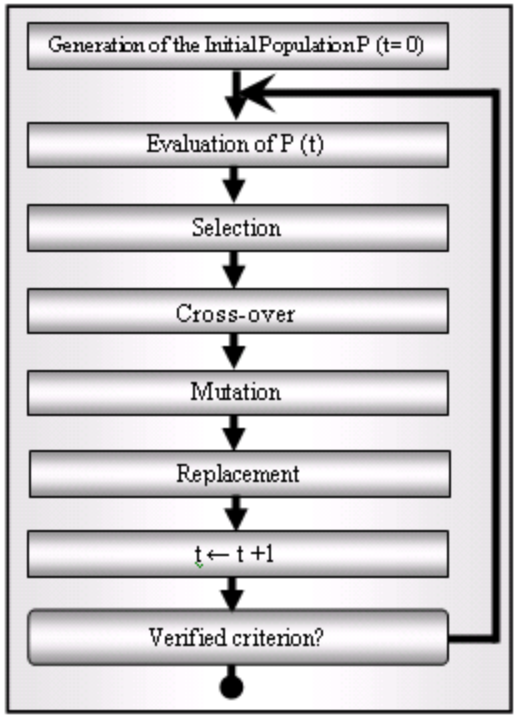

Figure 1. CGA structure

The simplest way for coding chromosomes is to represent them by binary strings. The initial population has to start with random chromosomes uniformly distributed over the entire search space. The next step is the evaluation operation. Its role is to mark the individuals of the population. After that, the individuals will be sorted according to their marks. The selection operation has as goal to elect some number of individuals to enable reproduction. The cross-over operation can be performed by exchanging some parts of selected individuals in random positions which leads to create a new set of chromosomes replacing the old ones. Before repeating the process, it is recommended to perform a mutation to correct stochastic errors to avoid a genetic drift and to ensure a genetic diversity in the population. It consists of changing some random positions of the individuals according to a small probability (typically between 1% and 0.1%).

3. Quantum genetic algorithm (QGA)

3.1 Quantum computing

In quantum computing, the smallest unit of information storage is the quantum bit (qubit) [4]. A qubit can be in the state 1, in the state 0 or in a superposition of both. The state of a qubit can be represented as [3]:

$$|\Psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle \tag{1}$$

Where $|0\rangle$ and $|1\rangle$ represent the values of classical bits 0 and 1 respectively; α and β are complex numbers satisfying:

$$|\alpha|^2 + |\beta|^2 = 1 \tag{2}$$

$|\alpha|^2$ is the probability where a qubit is in state 0 and $|\beta|^2$ represents the probability where a qubit is in state 1. A quantum register of m qubits can represent 2^m values simultaneously. However, when the 'measure' is taken, the superposition is destroyed and only one of the values becomes available for use. That's why we think that quantum computers can be used mainly in applications involving a choice among a multitude of alternatives.

In general, a quantum algorithm has less complexity than its classic equivalent algorithm through the concept of quantum superposition. Among the most famous quantum algorithms we quote Shore's algorithm for number factorization (1994) [14] and Grover's algorithm for research in a non sorted database (1996) [6]. Both algorithms have solved problems which their complexity was reduced.

3.2 QGA principles

QGAs are a combination between GA and quantum computing. They are mainly based on qubits and states superposition of quantum mechanics. Unlike the classical representation of chromosomes (binary string for instance), here they are represented by vectors of qubits (quantum register). Thus, a chromosome can represent the superposition of all possible states. The structure of a QGA is illustrated in figure 2 [4]:

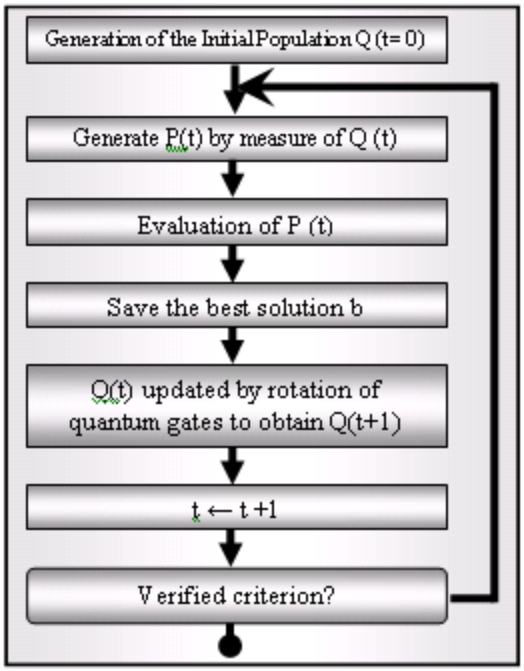

Figure 2. QGA structure

3.2.1 Structure of quantum chromosomes

A chromosome is simply a string of m qubits that forms a quantum register. Figure 3 shows the structure of a quantum chromosome.

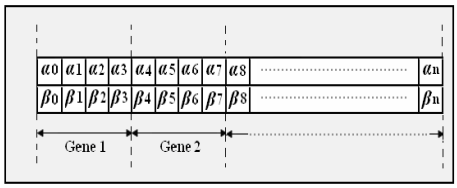

Figure 3. Quantum chromosome structure

3.2.2 Initializing the population

The easiest way to create the initial population is to initialize all the amplitudes of qubits by the value 1 / (2 ^ ½). This means that a chromosome represents all quantum superposition states with equal probability.

3.2.3 Evaluation of individuals

The role of this phase is quantifying the quality of each quantum chromosome in the population to make a reproduction. The evaluation is based on an objective function that corresponds to each individual, after measuring, an adaptation value. It permits to mark individuals in the population.

3.2.4 Quantum genetic operations

- **Measuring chromosomes**

In order to exploit effectively superposed states of qubits, we have to observe each qubit. This leads us to extract a classic chromosome. The aim is to enable the evaluation of each quantum chromosome.

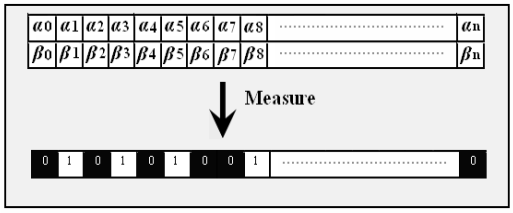

Figure 4. Measured chromosome

A simple way to implement this function is given by the following pseudo code:

```
Function measure ()
begin
     r := get r in [0,1] ;
     if (r > |α|²)
          return 1 ;
       else
          return 0 ;
     end if
end
```

- **Interference**

This operation allows modifying the amplitudes of individuals in order to improve performance. It mainly consists of moving the state of each qubit in the sense of the value of the best solution. This is useful for intensifying the search around the best solution. It can be performed using a unit transformation that allows a rotation whose angle is a function of the amplitudes (ai, bi) and the value of the corresponding bit in the reference solution. The value of the rotation angle δθ has to be chosen so that to avoid premature convergence. It is often empirically determined and its direction is determined as a function of the values of ai, bi and the value of the qubit located at the position i in the individual being modified [1].

- **Qubit rotation gates strategy**

The rotation of individual's amplitudes is performed by quantum gates. Quantum gates can also be designed in accordance with the present problem. The population Q (t) is updated with a quantum gates rotation of qubits constituting individuals. The rotation strategy adopted is given by the following formula:

$$\begin{bmatrix} \alpha_i^{t+1} \\ \beta_i^{t+1} \end{bmatrix} = \begin{bmatrix} \cos(\Delta\theta_i) & -\sin(\Delta\theta_i) \\ \sin(\Delta\theta_i) & \cos(\Delta\theta_i) \end{bmatrix} \begin{bmatrix} \alpha_i^{t} \\ \beta_i^{t} \end{bmatrix} \tag{3}$$

Where Δθi is the rotation angle of qubit quantum gate i of each quantum chromosome. It is often obtained from a lookup table to ensure convergence.

4. QGA *vs* CGA

4.1 Knapsack problem

The knapsack problem is one the most combinatorial algorithms. It can be described as selecting from among various items those which are most profitable, given that the knapsack has a limited capacity. There are many types of knapsack problem, so the simplest one is called 0/1 knapsack problem. It is described as: given a set of m items and a knapsack, select a subset of the items so as to maximize the profit f(x) [4] as shown in formula 4:

$$f(x) = \sum_{i=1}^{m} p_i x_i \tag{4}$$

Subject to:

$$f(x) = \sum_{i=1}^{m} w_i x_i \leq C \tag{5}$$

Where $x = (x_1, x_2, \ldots, x_m)$, x_i is 0 or 1, p_i and w_i are the profit and the weight of the ith item. C is the capacity of the knapsack.

4.1.1 Choosing parameters values

Since it was found that the difficulty of such problem is greatly affected by the correlation between profits and weights [3], three randomly generated sets of data are considered [17]:

- **Uncorrelated**

  w_i = (uniformly) random([1..v])
  p_i = (uniformly) random([1..v])

- **Weakly correlated**

  w_i = (uniformly) random([1..v])
  $p_i = w_i$ + (uniformly) random([-r..+r])

- **Strongly correlated**

  w_i = (uniformly) random([1..v])
  $p_i = w_i + r$

Higher correlation implies smaller value of the difference:

$$\max_{i=1..m} \{p_i / w_i\} - \min_{i=1..m} \{p_i / w_i\};$$

As reported in [3], higher correlation problems have higher expected difficulty.

The knapsack capacity can be set according two types (again, following a suggestion from [3]):

- **Restrictive knapsack capacity (C1)**

The knapsack capacity C= 2v. In this case, the optimal solution contains very few items [17].

- **Average knapsack capacity (C2)**

The knapsack capacity is computed as fellow:

$$f(x) = \tfrac{1}{2} \sum_{i=1}^{m} w_i \tag{6}$$

In this case, the optimal solution contains about half of the items [17].

4.1.2 Solving knapsack problem with conventional GA (CGA)

There are three types of conventional GA algorithms [17]: algorithms based on penalty functions, algorithms based on repair methods and algorithms based on decoders.

In algorithms based on penalty functions, each solution is coded as a binary string of the length m representing a chromosome x to the problem. The profit f(x) of each chromosome is computed as shown in formula 7:

$$f(x) = \sum_{i=1}^{m} p_i x_i - Pen(x) \tag{7}$$

Where Pen(x) is a penalty function. We consider here the three types of penalties discussed in [17]: logarithmic penalty, linear penalty, and quadratic penalty:ρ

$$\begin{aligned} Pen_1(x) &= log_2(1 + \rho(\sum_{i=1}^{m} p_i x_i - C)) \\ Pen_2(x) &= \rho(\sum_{i=1}^{m} p_i x_i - C) \\ Pen_3(x) &= (\rho(\sum_{i=1}^{m} p_i x_i - C))^2 \end{aligned} \tag{8}$$

Where ρ is $\max_{i=1..m} \{p_i / w_i\}$. The penalty function Pen(x) is zero for all feasible solutions x, i.e., solutions that:

$$f(x) = \sum_{i=1}^{m} w_i x_i \leq C \tag{9}$$

And it is greater than zero otherwise.

In algorithms based on repair methods, the profit f(x) of each chromosome is computed as shown in formula 10:

$$f(x) = \sum_{i=1}^{m} p_i x_i' \tag{10}$$

Where x' is a repaired vector of the original vector x. Original chromosomes are replaced with a 5% probability in the experiment (it has been proved that that 5% is the most appropriate replacement percentage). We have used the two repair algorithms mentioned in [17]. The repair algorithms differ only in selection procedure, which chooses an item for removal from the knapsack:

- **Random repair (Rep1):** in this case, a random element is removed from the knapsack.
- **Greedy repair (Rep2):** All items in the knapsack are sorted in the decreasing order of their profit to weight ratios. The last item is always chosen for deletion.

We will not tackle here the third method (decoder based algorithms), because the chromosome representation is based on integers while quantum chromosomes can use only qubit representation.

4.2 Tests and results

In all our experiments we have coded solutions as binary strings of the length m for CGA and as qubit strings of the length m for QGA. The length of both strings is the same as the number of items. For CGA the ith item is added to the knapsack if and only if the ith element in the binary string is 1. Similarly for CGA, the ith item is added to the knapsack with a probability of $|\beta_i|^2$ where $|\beta_i|$ is the amplitude of the ith qubit in the qubit string. Before presenting our experimental results, we will announce the parameters of both algorithms and knapsack problem parameters.

5. CGA parameters

Parameter	Value
Cross-over probability	50 %
Mutation probability	1 %
Population size	100

Table 1. List of CGA parameters

5.1 QGA parameters

The population size was fixed to be 100. The amplitudes of the individuals are updated by a rotation of quantum gates according to the look-up table 2.

xi bi	f (x) > f(b)	Δθi	s (ai bi)			
			ai.bi > 0	ai.bi < 0	ai= 0	bi= 0
0 0	0	0.001π	-	+	±	±
0 0	1	0.001π	-	+	±	±
0 1	0	0.08π	-	+	±	±
0 1	1	0.001π	-	+	±	±
1 0	0	0.08π	+	-	±	±
1 0	1	0.001π	+	-	±	±
1 1	0	0.001π	+	-	±	±
1 1	1	0.001π	+	-	±	±

Table 2. Look-up table for quantum gates rotation

xi and bi are the i-th bits of x and b (the best solution), respectively. f is the fitness function and s (ai bi) is the sign of the rotation angle θi. According to the lookup table, one can easily remark that this strategy improves, for each individual, the amplitudes of qubits that are bad according to an angle $\delta\theta 1 = 0.08\pi$ while it decreases, those that are good according to an angle $\delta\theta 2 = 0.001\pi$. The amplitude values were empirically determined. The modification of the amplitudes of qubits is done according to the signs of the amplitudes, the best solution and the solution extracted by the individual container. It is natural that $\delta\theta 1 >> \delta\theta 2$ because decreasing amplitudes serves only to correct stochastic errors to avoid a genetic drift and to ensure a genetic diversity in the population.

5.2 Knapsack problem parameters

Parameter	Value
V	10
R	5
Repairing replacement percentage	5%

Table 3. List of knapsack problem parameters

5.3 Experimental results

We have executed both algorithms (CGA & QGA) over 50 runs for all possible cases, at each one the concerned algorithm was iterated for a maximum number = 500 generations. Table 4 shows the experimental results of the knapsack problem with 100, 250 and 500 items.

6. Parallelization of QGA

6.1 Evolving Cellular Automata (CA) by QGA

In this section we propose to evolve CA by QGA in a parallel environment to get out some computational tasks and extract some computational abilities of QGA to perform processes in an effective and efficient manner. Indeed, the difficulty of designing a CA to present a specific behavior or to perform a particular task has limited their applications; one possible solution is the automation of the design process (programming) to enhance the CA's viability where transition rules should be carefully produced by hand [8] [13]. In the literature, most of papers that deal automatic learning of CA rules focus on one technique: Evolving CAs by evolutionary algorithms what is known as evolutionary cellular automata. The beginning was with Packard, then Mitchell and al where a GA is used to evolve CAs to solve the density classification problem [8] [12]. Other experiments have been carried out later, such as those made by Sipper where he used a GA to evolve another kind of CAs so-called non-uniform (where cells may have different rules) to perform the density classification task [11], and Sapin who used evolutionary algorithms to discover universal CAs [2]. Some researchers have also evolved CAs to perform some image processing tasks [7] [13].

Correl	No. of Items	Capacity type		method									
				CGA					QGA				
				Pen1	Pen2	Pen3	Rep1	Rep2	Pen1	Pen2	Pen3	Rep1	Rep2
none	100	C1	m	-	*	*	71.0	71.2	-	*	*	69.6	71.2
			b	-	*	*	92.5	87.6	-	*	*	92.7	87.6
			w	-	*	*	53.8	51.7	-	*	*	53.4	51.7
		C2	m	-	376.1	384.5	416.8	417.1	-	378.0	386.6	417.0	417.0
			b	-	422.5	430.4	450.2	436.9	-	424.8	433.5	449.4	436.9
			w	-	336.5	337.0	389.0	387.6	-	338.5	338.3	389.1	387.2
	250	C1	m	-	*	*	69.0	91.6	-	*	*	65.0	91.7
			b	-	*	*	83.7	125.7	-	*	*	80.6	125.9
			w	-	*	*	57.3	75.1	-	*	*	52.8	75.1
		C2	m	-	978.9	975.1	999.6	1024.4	-	1034.5	1035.6	1043.7	1026.5
			b	-	1020.4	1010.7	1066.3	1065.7	-	1077.1	1078.0	1101.5	1068.1
			w	-	938.0	926.4	944.6	966.2	-	961.4	985.9	991.9	968.5
	500	C1	m	-	*	*	68.7	141.5	-	*	*	66.0	142.2
			b	-	*	*	84.0	157.4	-	*	*	80.6	158.5
			w	-	*	*	55.0	93.7	-	*	*	54.6	93.7
		C2	m	-	1771.6	1812.7	1889.0	2016.7	-	1889.3	1967.8	2039.3	2071.3
			b	-	1897.4	1894.4	1946.5	2101.1	-	2061.2	2076.8	2103.6	2174.2
			w	-	1670.4	1748.5	1796.2	1925.6	-	1754.8	1873.6	1928.3	1968.9
weak	100	C1	m	-	*	*	47.7	44.4	-	*	*	44.8	44.4
			b	-	*	*	64.9	56.8	-	*	*	64.9	56.6
			w	-	*	*	36.9	32.3	-	*	*	36.9	32.3
		C2	m	-	397.5	396.2	398.2	397.0	-	398.5	398.1	398.1	397.0
			b	-	430.0	429.3	422.2	435.5	-	430.0	428.3	422.2	435.5
			w	-	360.4	359.2	365.9	357.2	-	361.6	361.1	365.1	357.2
	250	C1	m	-	*	*	46.2	67.3	-	*	*	44.5	67.3
			b	-	*	*	52.8	78.9	-	*	*	51.4	78.9
			w	-	*	*	40.3	51.1	-	*	*	38.9	51.1
		C2	m	-	940.9	924.4	950.5	991.1	-	991.5	985.0	987.1	991.3
			b	-	981.5	959.9	1016.9	1058.9	-	1033.3	1024.4	1056.0	1059.4
			w	-	874.1	886.6	870.5	944.1	-	932.1	944.5	912.8	944.8
	500	C1	m	-	*	*	40.9	83.3	-	*	*	40.8	83.4
			b	-	*	*	46.5	97.7	-	*	*	45.5	97.7
			w	-	*	*	37.0	50.6	-	*	*	36.4	50.6
		C2	m	-	1767.2	1747.1	1817.2	1969.8	-	1942.1	1933.2	1957.5	1993.2
			b	-	1849.6	1798.2	1915.9	2070.6	-	2000.2	1980.0	2044.9	2097.9
			w	-	1708.3	1684.1	1742.8	1901.8	-	1880.8	1887.1	1881.2	1931.3
Strong	100	C1	m	-	*	*	79.9	81.0	-	*	*	79.5	81.1
			b	-	*	*	90.0	90.0	-	*	*	90.0	90.0
			w	-	*	*	69.9	70.0	-	*	*	65.0	70.0
		C2	m	-	607.2	603.3	611.5	612.6	-	608.8	608.0	611.2	612.5
			b	-	619.8	614.6	628.4	628.0	-	622.2	619.6	628.5	628.0
			w	-	590.2	592.7	582.8	600.7	-	590.4	593.5	583.4	600.7
	250	C1	m	-	*	*	70.8	95.0	-	*	*	68.6	95.6
			b	-	*	*	79.8	105.0	-	*	*	75.0	105.0
			w	-	*	*	64.7	85.0	-	*	*	59.9	85.0
		C2	m	-	1468.9	1460.2	1493.7	1528.7	-	1521.3	1522.9	1527.8	1533.4
			b	-	1492.9	1482.8	1521.7	1562.5	-	1543.5	1545.8	1550.2	1567.5
			w	-	1427.3	1434.4	1454.9	1483.3	-	1495.4	1494.9	1490.6	1488.9
	500	C1	m	-	*	*	68.1	102.4	-	*	*	67.3	103.2
			b	-	*	*	74.9	110.0	-	*	*	74.2	110.0
			w	-	*	*	64.6	95.0	-	*	*	59.6	95.0
		C2	m	-	2864.0	2846.6	2914.6	3003.3	-	3005.2	2999.0	3026.7	3060.2
			b	-	2893.3	2889.1	2951.4	3033.9	-	3028.8	3032.9	3061.5	3088.7
			w	-	2821.6	2806.9	2886.4	2968.6	-	2967.6	2966.9	3000.8	3025.0

Table 4. Experimental results of the knapsack problem

b, **m**, and **w** mean **best**, **mean**, and **worst**, respectively.
'-' means that an experiment did not made in this case.
'*' means that no valid solution has been found within given constraints.

6.1.1 CA Principles

Cellular automata are dynamic systems whose main characteristic is to depend on rules that can be very simple but whose global behavior can be very complex. Their simplicity and their rigorous mathematical model have strongly encouraged their application in many fields. A CA is a lattice of N cells where each cell is in one of K possible states at time t. Each cell follows the same rule to update its state. The state s of a cell at time t +1 depends on its state and the states of a number of neighboring cells at time t. The CA starts with an initial configuration (CI) of cells and at each time step the states of all cells in the lattice are updated in a synchronous manner. A transition rule Φ can be expressed in a table called "look-up table" which lists for each local neighborhood, the state's update of the neighborhood's central cell [9]. For example, if one considers a one-dimensional CA with two states and a neighborhood radius r=1, the look-up table for updating cells could be given as indicated in table 5.

N	000	001	010	011	100	101	110	111
S	0	1	1	1	0	0	0	1

Table 5. Neighborhood look-up table

6.1.2 Solving the density classification problem by QGA

We examine in this study CAs where: cells can have only two possible states (0,1), evolved configurations are 1-D lattices, transition rules take into account the three direct left neighbors of a cell and its three neighbors in the right (the concerned cell will be included), the radius of the neighborhood is three (r = 3). The neighborhood η of each cell is composed of seven cells in total. Each cell can have 128 different configurations for η. The transition rule must associate to each configuration of η the value to be taken by the concerned cell in the next time. Therefore, the space of all CAs should include 2 ^ 128 different CA.

To demonstrate the power of QGA for evolving CAs, we have considered the most studied problem type: the density classification problem ρc = ½. For this task, the CA decides for each initial configuration of black and white cells, if there are more black cells than white ones. When there are more black cells, the CA should achieve a pattern where all cells are black. We want to use a QGA to try to obtain CAs that are able to perform this task. The QGA must also stop when the best individual is able to perform the given task or the algorithm has been iterated for some number of generations. This is a non-trivial task for a CA, since the existence of a majority of black cells is a global property of a configuration, while each cell of the automaton can only communicate locally. It was proved that this task is insolvable for CAs [10], i.e. there is no CA that can perform this task for all possible configurations. This task can then be considered as a good criterion for measuring the viability of QGA to explore CA search spaces to deduce rules that cover a wide range of configurations.

- **Coding transition rules**

Each CA is coded as a quantum register of 128 qubits where each qubit is associated with one of 128 possible configurations of η. Figure 5 illustrates the structure of each quantum chromosome belonging to the population.

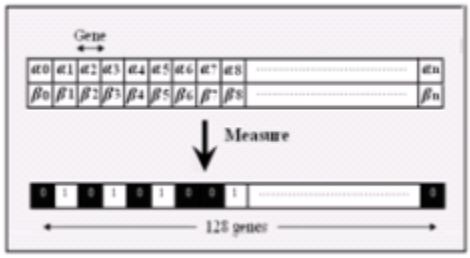

Figure 5. CA quantum chromosome structure

- **Initializing the population**

The tests for each rule transition were performed on configurations of length N = 149, with respect to periodic boundaries conditions. The population was composed of 100 individuals, which was in fact the size of the population used in the test of Mitchell [8]. The initial population is generated so that all quantum amplitudes of chromosomes were initialized by 1 / (2 ^ ½). For each quantum chromosome is associated a set of 300 randomly generated initial configurations but forced to be uniformly distributed over ρ Є [0.0, 1.0]. At each generation, a new set of 300 initial configurations is generated.

- **Evaluation of individuals**

The evaluation of each quantum chromosome is performed, after measure operation, by evolving its whole initial configurations using the extracted chromosome for M iterations. For each individual is attributed a mark which is the fraction of cells for which the final state is correct.

- **Interference**

Once the evaluation of the population is performed, the CA having the highest adaptation value is determined. The amplitudes of the individuals are updated by a rotation of quantum gates according to the look-up table 6.

xi bi	f (x) > f(b)	Δθi	s (ai bi)			
			ai.bi > 0	ai.bi < 0	ai= 0	bi= 0
0 0	0	0.004π	-	+	±	±
0 0	1	0.004π	-	+	±	±
0 1	0	0.08π	-	+	±	±
0 1	1	0.004π	-	+	±	±
1 0	0	0.08π	+	-	±	±
1 0	1	0.004π	+	-	±	±
1 1	0	0.004π	+	-	±	±
1 1	1	0.004π	+	-	±	±

Table 6. Look-up table for quantum gates rotation.

One can easily observe that this strategy is very similar to that shown in table 2 with a small difference at the values of δθ1 and δθ2 (0.08π and 0.004π respectively).

6.2 Parallelizing the QGA process

It is possible for a QGA to evaluate and to interfere many chromosomes in parallel. It seems that master / slave model can be very adequate to realize this parallelization. Then, if we consider (m+1) connected machines (LAN for example) we can easily construct a parallel version of QGA by choosing randomly a site i that plays the role of a coordinator for synchronization of sites that execute the evaluation operation. Each site that finished the operation of evaluation should inform the site i, which reacts by determining the best solution, then it sends it to various sites to perform the interference operation in parallel. So, the m remaining sites are slaves carrying out the evaluation and interference tasks in a parallel manner. The entire population will be divided into m subpopulations. Each one will be assigned to a slave site.

6.3 Tests and results

6.3.1 Efficiency measurement

We have executed our algorithm for 100 generations. Each CA in the population was iterated over M = 150 iterations. We have compared our best fitness obtained with the best that exists in the literature. Table 7 shows the obtained results.

Approach	Fitness
QGA	94.18 %
GA (Best fitness in literature [Mitchell and al 1993])	93% - 95 %

Table 7. Fitness Table.

6.3.2 Performance measurement

Our parallel algorithm was run on a local network of 10 sites: a master site and nine slaves. The hardware and software configuration of each site was as follows: Intel Pentium 4 (3.0 GHz), 512 MB of memory, Windows XP OS, Java Programming language (JDK 1.5.0). Three tests were performed for each one the number of slave sites has been varied: measurement of the average duration of communication between the different slave sites and the master site, the average duration of a generation and the achievable acceleration. The size of manipulated populations in function of used sites' number is shown in table 8.

Nb of slave sites	2	3	4	5	6	7	8	9
Size of populations	100	99	100	100	96	98	96	99
size of each sub- population	50	33	25	20	16	14	12	11

Table 8. Size of manipulated populations

Indeed, we must assign to each processor the same amount of information to process to ensure a better exploitation of available parallelism. The obtained results for each test are mentioned in figures 6, 7, 8.

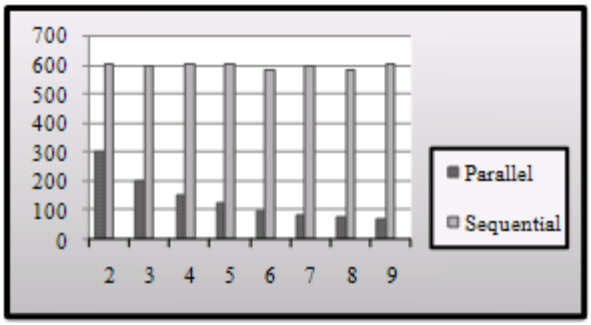

Figure 6. Variations of average time for one generation in function of sites' number (Time is measured in seconds (s))

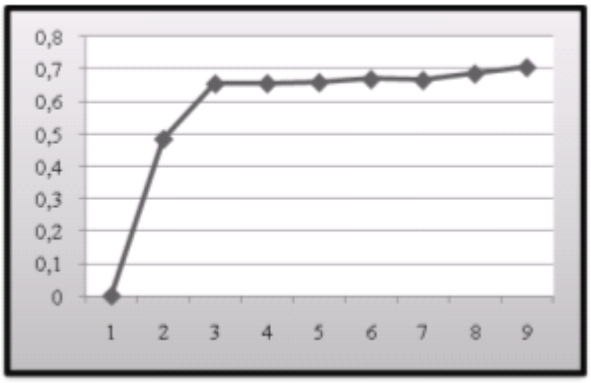

Figure 7. Variations of communication average time for one generation in function of sites' number (Time is measured in seconds (s))

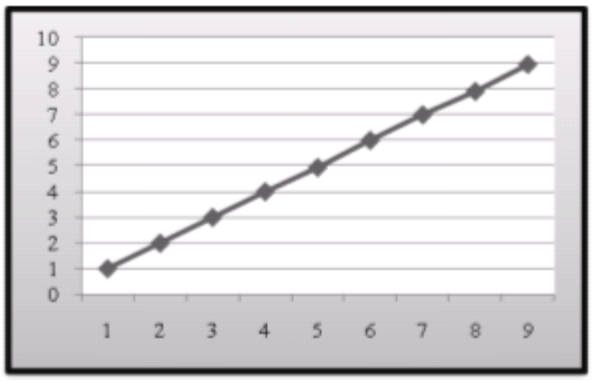

Figure 8. Variations of achieved acceleration in function of sites' number

The acceleration is computed by the following formula:

$$A = \frac{\text{time in sequential execution}}{\text{time in parallel execution.}} \tag{11}$$

7. Discussion

Table 4 shows the experimental results of the knapsack problem where the number of items was 100, 250 and 500. The hardware and software configuration was as follows: Intel Pentium 4 (3.4 GHz), 1 Go MB of memory, Windows XP OS, Java Programming language (JDK 1.5.0).

In the case of 100 items, both algorithms were equivalent for all problem instances. This is because the number of items is so small (we have only 2 ^ 100 possibility). However, augmenting the number of items (500 items where we have 2 ^ 500 possibility) leads QGA to behave better than CGA especially for penalty based algorithms, and this in all problem solution variants.

By using repairing methods, table 4 shows that for a small number of items the profits are approximately close. In the case of Rep2 elements are selectively removed from the knapsack, in contrast to Rep1 where elements are randomly removed. In fact, using repairing methods influences directly the evolution process. For instance, we can easily compare the best results obtained by penalty based algorithms and those obtained using repairing algorithms. Moreover, we can easily observe that when the capacity of the knapsack was equal to C1 = 2v (a very small capacity) no solution was found in the case of penalty methods which proves that repairing methods have a great effect. In the other hand, varying the correlation between profits and weights from none to strong (augmenting difficulty level) makes QGA to behave better and better than CGA (we can easily verify best fitness from table 4). To check theses issues we have extended the number of items to 1000, the obtained penalty fitness values are presented in table 8. So we can then judge that QGA's performance is higher than the one of CGA when taking into account the evolution process on its own.

Now we will tackle the density classification problem. Concerning efficiency aspect, table 7 shows the similarity between the fitness obtained by QGA and the one obtained by GA. The density classification problem is generally regarded as a very difficult problem to solve by CAs; it can be a good criterion to measure the power of exploring CAs spaces. It is clear that for this task, our algorithm was able to deduce appropriate rules and just like a standard GA. This proves that the QGA and GA can be at least equivalent and we may therefore judge that QGA can be a powerful tool for exploring CA search spaces.

Now we analyze the performance aspect, according to figure 6, we see clearly the difference between a generation average time executed in parallel and a generation average time executed in sequence: the execution time is reduced when the sites' number is increased which means that the performance increases, as proof, we just check figure 8 where the acceleration increases linearly with the number of sites and even tends to the number of processors. One possible interpretation of these results: the communication phase time is very low (of the order of milliseconds). Indeed, that is what we observe in figure 7 which shows that the communication average time increases very slowly when the number of used processors is increased. By comparing communication average time with the generation average time (parallel execution), we notice that communication time << generation average time, and thus guarantees a highly efficient performance.

One can ask about interest by QGA compared to CGA. Quantum algorithms have generally the ability to minimize the complexity of equivalent algorithms that run on classic computers. We can make a simple comparison between the global complexity of QGA and the one of CGA to estimate the reduction in complexity we can achieve. Starting with QGA, the global complexity is of the order of O(N), N is the size of the population (Evaluation + Interference).

For a standard GA, the global complexity is often of the order of $O(N^2)$ (Evaluation + Selection + Cross-over + Mutation). Therefore, we believe that this result is very interesting because the complexity here has been reduced to become linear. Indeed, one can imagine what happen if we consider a very large population of chromosomes; it will be very useful to use QGA instead of GA. Another very interesting feature for using QGA is the possibility of performing parallel execution. According to the structure of QGA shown in Figure 2, it appears that they can support parallel processing in particular the evaluation and interference phases where we can process multiple quantum chromosomes simultaneously. We can therefore conclude that QGA can be a very powerful tool for exploring search spaces efficiently and effectively.

No. of Items	Correl	Capacity type		method					
				CGA			QGA		
				Pen1	Pen2	Pen3	Pen1	Pen2	Pen3
1000	none	C1	m	-	*	*	-	*	*
			b	-	*	*	-	*	*
			w	-	*	*	-	*	*
		C2	m	-	3347.8	3345.8	-	3650.1	3658.0
			b	-	3503.5	3474.0	-	3922.7	3906.7
			w	-	3245.6	3230.7	-	3476.7	3487.3
	weak	C1	m	-	*	*	-	*	*
			b	-	*	*	-	*	*
			w	-	*	*	-	*	*
		C2	m	-	3319.4	3290.9	-	3715.9	3689.0
			b	-	3403.6	3386.5	-	3813.5	3784.9
			w	-	3230.4	3178.7	-	3611.3	3581.2
	strong	C1	m	-	*	*	-	*	*
			b	-	*	*	-	*	*
			w	-	*	*	-	*	*
		C2	m	-	5606.4	5567.7	-	5894.2	5865.4
			b	-	5664.6	5616.1	-	5933.5	5906.7
			w	-	5560.3	5515.7	-	5837.6	5816.3

Table 8. Extended experimental results of knapsack problem (1000 items)

8. Conclusion

In this work we have demonstrated the power of QGA for exploring search spaces. We have considered the density classification problem which is a very difficult problem to solve by a CA and the knapsack problem which is one of the most combinatorial problems as objective problems. The experimental results showed that QGA can be a very powerful tool for exploring search spaces while preserving the relation efficiency / performance. Our future work will focus on parallel execution of QGA to check the satisfaction of the scalability property in order to extract the potential parallelism offered by this technique. Another perspective of this work is to study different QGA strategies to dominate the effect of choosing rotation gate angles because they are often empirically determined.

References

[1] Layeb, A., Saidouni, D.E. (2007). Quantum Genetic Algorithm for Binary Decision Diagram Ordering Problem, *IJCSNS International Journal of Computer Science and Network Security*, 7 (9).

[2] Sapin, E., Bailleux, O., Chabrier, J.J., Collet, P. (2004). A New Universal Cellular Automaton Discovered by Evolutionary Algorithms, Gecco2004. Lecture Notes in Computer Science, 3102.175–187.

[3] McDonnell, J.R., Reynolds, R.G., Fogel, D.B. (1995). (Editors), Proceedings of the Fourth Annual Conference on Evolutionary Programming, The MIT Press.

[4] Han, D.B., Kim, J.H (2000). Genetic Quantum Algorithm and Its Application to Combinatorial Optimization Problem, *In*: IEEE Proc. Of the 2000 Congress on Evolutionary Computation, p. 1354-1360.

[5] Han, K.. Park, K Lee, C.H. Kim, J.hH. (2001). Parallel Quantum-inspired Genetic Algorithm for Combinatorial Optimization Problem, *In*: IEEE Proc. Of the 2001 Congress of Evolutionary Computation, on page(s): 1422-1429 v. 2.

[6] Grover, L.K. (1996). A Fast Quantum Mechanical Algorithm for Database Search, Proceedings, 28th Annual ACM Symposium on the Theory of Computing, ACM Press, p. 212-221.

[7] Batouche, M., Meshoul, S., Abbassene, A. (2006). On solving edge detection by emergence, *In*: The 19th International Conference on Industrial Engineering & Other Applications of Applied Intelligent Systems (IEA/AIE'06), Annecy, (France), Lecture Notes in Artificial Intelligence (LNAI 4031), p.800-808.

[8] Mitchell, M., Hraber, P.T., Crutchfield, J.P. (1993). Revisiting the Edge of Chaos: Evolving Cellular Automata to Perform Computations, Complex Systems, 7. 89-130.

[9] Mitchell, M., Hraber, P.T., Crutchfield, J.P. (1994). Evolving cellular automata to perform computation: Mechanisms and impedients, *Phys. D,* 75, p.361–391

[10] Land, M., Belew, R. (1995). NO Two-State CA for Density Classification Exists, *Physical Review Letters* 74, 5148.
[11] Sipper, M. (1996), co-evolving Non-Uniform Cellular Automata to Perform Computations, *Physica D*, 92, p. 193-208.
[12] Packard, N.H. (1988). Adaptation toward the edge of chaos. *In*: J.A.S.Kelso, A.J.Mandell, and M.F. Shlesinger, editors, Dynamic Patterns in Complex Systems, p. 293–301, Singapore, World Scientific.
[13] Rosin, P.L. (2006). Training Cellular Automata for Image Processing, *IEEE Transactions on Image Processing,* 15 (7) 2076 - 2087.
[14] Shor, P.W. (1994). Algorithms for Quantum Computation: Discrete Logarithms and Factoring, *In*: Proceedings of the 35th Annual Symposium on the Foundation of Computer Sciences, p. 20-22.
[15] Ma, S. Jin, W. (2007). A New Parallel Quantum Genetic Algorithm with Probability-Gate and Its Probability Analysis, *In*: International Conference on Intelligent Systems and Knowledge Engineering ISKE.
[16] Laboudi, P.L., Chikhi, S. (2009). Evolution d'Automates Cellulaires par Algorithmes Génétiques Quantiques, Conférence Internationale sur l'Informatique et ses Applications (CIIA'09).
[17] Michalewicz, Z. (1999), Genetic Algorithms + Data Structures = Evolution Programs, Springer-Verlag, 3rd, revised and extended edition.