QUANTUM CIRCUIT DESIGN BY MEANS OF GENETIC PROGRAMMING⋆

ANDREI BĂUTU[1], ELENA BĂUTU[2]

[1] "Mircea cel Bătrân" Naval Academy, Constantza, 900218, Romania, abautu@anmb.ro
[2] "Ovidius" University, Constantza, 900527, Romania, erogojina@univ-ovidius.ro

Received September 12, 2006

Research in quantum technology has shown that quantum computers can provide dramatic advantages over classical computers for some problems. The efficiency of quantum computing is considered to become so significant that the study of quantum algorithms has attracted widespread interest. Development of quantum algorithms and circuits is difficult for a human researcher, so automatic induction of computer programs by means of genetic programming, which uses almost no auxiliary information on the search space, proved to be useful in generating new quantum algorithms. This approach takes advantage of the intrinsic parallelism of the genetic algorithm and quantum computing parallelism. The paper begins with a brief review on some basic concepts in genetic algorithms and quantum computation. Next, it describes an application of genetic programming for evolving quantum computing circuits.

Key words: quantum gates, quantum algorithms, genetic programming.

1. INTRODUCTION

Quantum computing is a new computational paradigm that has proven efficient in solving problems that are hard for conventional paradigms, such as factoring large integers in polynomial time [1]. The discovery and optimization of quantum algorithms by human researchers can be quite a frustrating experience, but automatic searching techniques can be applied to ease this task. Some authors tackled the problem of quantum circuit design with nature-inspired algorithms. Genetic programming is such a nature inspired meta-heuristic that has been applied to various artificial intelligence problems with great success. We present in this paper a novel genetic programming scheme based on S-expressions and adapted to the particular features of quantum circuits. The experimental results demonstrate the effectiveness and the applicability of this approach. Section 2 presents the basic concepts of the genetic programming paradigm. Section 3 is

⋆ Paper presented at the 7th International Balkan Workshop on Applied Physics, 5–7 July 2006, Constanța, Romania.

Rom. Journ. Phys., Vol. 52, Nos. 5–7, P. 697–704, Bucharest, 2007

a brief introduction to quantum computation. Section 4 describes the application of genetic programming for evolving quantum computing circuits, while conclusions and future research directions are presented in the last section.

2. GENETIC PROGRAMMING

In his famous 1950 paper [2], Turing suggested one of three directions for artificial intelligence research should be automatic program design by means of an evolutionary inspired approach. He was the first to envision evolutionary techniques that evolve computer programs for problem solving. In Turing's approach, an initial solution is transformed over time using mutations; each mutation is judged by a human expert who decides which mutations are accepted based of their effects. Turing never implemented his technique, but his belief was that programs could learn intelligence from their human judges.

Among the first implementations of Turing's ideas was Holland's Genetic Algorithm (GA) [3]. GA maintains a set of candidate solutions and evolves them through time. Holland proposed that the merits of a solution be automatically evaluated by a fitness function, instead of the human expert. The evolution process involves a kind of natural selection and genetically inspired operators of mutation and crossover. The chances of each individual to survive into the next generation are proportionate to its fitness.

In the early 1990s, Koza [4] introduced Genetic Programming (GP) as a variation of GA. GP is an evolutionary algorithm that uses a functional encoding of solutions – initially, the candidate solutions were encoded by Lisp S-expressions. The individuals are computer programs encoded as syntax trees with varying size and shape. The internal nodes of the tree are labeled with functional symbols. The number of child branches of each node matches the arity of the function that labels it. Nodes on the frontier of the tree are labeled with constants or variables (they are considered to be 0-arity nodes). For example, the expression (a + (3-a)/5) can be represented by the S-expression (+ a(/ (– (3 a)) 5)) which can be represented as the syntax tree:

The basic GP algorithm is as follows:

1. create the initial population of random individuals;
2. evolve individuals with genetic operators;

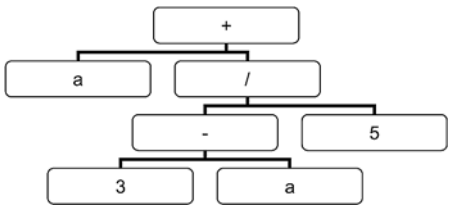

Fig. 1 – Parse tree for the S-expression (+ a (/ (– (3 a)) 5)).

3. evaluate individuals over a set of fitness cases;
4. select individuals in the next generation;
5. go to 2 if termination criterion is not met.

Usually, the termination criterion is set as a maximum number of generations the algorithm is allowed to run. Alternative criteria include Cauchy like conditions – the algorithm stops if the solution has not improved significantly over a given number of generations. The solution designated by a run of the GP algorithm is the best individual throughout the generations.

Genetic programming uses an evaluation function to determine how well an individual solves a given problem. The fitness value allows the selection scheme to distinguish between individuals of different qualities. In order to compute the fitness values, individuals are applied to a fitness case, or a set of fitness cases (depending on the problem). The score on the fitness cases represents the basis for the calculation of the final fitness function value.

3. QUANTUM COMPUTATION

The basic elements of quantum computing are quantum bits, quantum registers, quantum gates and quantum circuits. A building block of classical computational devices is a two-state system, *e.g.* {0, 1}. Any system with a finite set of discrete stable states, with controlled transitions between them is sufficient.

The basic information unit in the quantum computer is a quantum bit, for short *qubit*. A quantum bit represents an abstract quantum mechanical system with two distinct basis states. Hence, a qubit exists in a superposition of the two eigenstates denoted by $|0\rangle$ and $|1\rangle$. In general, a qubit is represented in the form $|\psi\rangle = a_0|0\rangle + a_1|1\rangle$, where a_0 and a_1 are complex numbers called probability amplitudes of the basis states, such that $|a_0|^2 + |a_1|^2 = 1$.

A quantum system with n qubits is described by $N = 2^n$ independent states, obtained as the tensor product of the Hilbert space associated with each qubit. The physical realization of a quantum system with n qubits is a quantum register. The state of a quantum register can be expanded as a superposition of the $N = 2^n$ basis states:

$$|\psi\rangle = \sum_{i=0}^{N-1} a_i |i\rangle, \qquad \sum_{i=0}^{N-1} |a_i|^2 = 1 \tag{1}$$

A state that can be written as a tensor product of qubits belongs to the tensor product of the Hilbert spaces associated to the qubits. However, the tensor product of the Hilbert spaces associated to the qubits contains states that can not

be represented as tensor products of basic qubits states. They are called entangled states, the foundations for quantum teleportation. A well known entangled state is the EPR (Einstein, Podolsky, Rosen) pair from (2). For this system, if qubit 1 is measured as $|1\rangle$ (with 50% probability) then measurement of qubit 0 will result in $|1\rangle$ (with 100% probability).

$$|\psi\rangle = \frac{1}{\sqrt{2}}\left(|0\rangle_1 \otimes |0\rangle_2 + |1\rangle_1 \otimes |1\rangle_2\right) = \frac{1}{\sqrt{2}}\left(|00\rangle \otimes |11\rangle\right) \tag{2}$$

The unobserved evolution through time of a quantum computer (*i.e.* the system does not interact with its environment) is governed by unitary transformations. A quantum computer can be described by a set of quantum gates. Each gate is a unitary linear operation described by a unitary matrix U. Therefore the evolution of a quantum computer can be described as an ordered sequence of unitary matrices applied to the column vector of a quantum register Q. The result of applying U to Q is given by the matrix vector multiplication UQ.

Among the most commonly used quantum gates are NOT, Walsh-Hadamard (H), Square NOT (SRN), Controlled NOT (CNOT), Simple Rotation (SR), and Swap (Sw). These gates are presented in Table 1.

Table 1

Frequently used unary and binary quantum gates

Gate	Unitary matrix	Symbol
NOT	$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$	
Hadamard	$\frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$	
Square NOT	$\frac{1}{\sqrt{2}}\begin{pmatrix} 1 & -1 \\ 1 & 1 \end{pmatrix}$	
Controlled NOT	$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$	
Simple rotation	$\begin{pmatrix} \cos\phi & \sin\phi \\ -\sin\phi & \cos\phi \end{pmatrix}$	
Swap	$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$	

4. QUANTUM CIRCUIT DESIGN BY MEANS OF GP

4.1. PREVIOUS WORK

In the field of applying evolutionary algorithms to quantum computation there are two major directions: designing quantum inspired evolutionary algorithms [5–7] and designing quantum computers by means of evolutionary algorithms. The research in the area of quantum circuit design focuses mainly on using GP for automatically discovering quantum algorithms. Williams [8–9] uses evolution of quantum circuits through GP to find alternative circuits for a quantum algorithm given by its unitary matrix. Spector [10–11] proposed different GP schemes for discovering quantum circuits based on their expected outcome (*e.g.* the AND-OR query problem – to find the result of a known boolean function over values returned by a black box function).

4.2. OUR APPROACH

We present a new scheme for genetic programming which uses the standard GP representation in the form of S-expressions, adapted to the particular features of quantum circuits. For this new encoding we present a corresponding set of genetic operators. In the following, we will refer to a quantum circuit as a sequence of quantum gates (*i.e.* unitary matrices) operating on a quantum register (*i.e.* N-dimensional column vector). Besides its target qubits, some gates require additional parameters, like control qubits and/or other parameters.

In our scheme, a GP individual encodes a single quantum circuit, *i.e.* a list of quantum gates. The gates are encoded in the form *sn[p]*. The symbol s represents the type of the gate from a given set of gate types, the integer n encodes the target qubits of the gate $\left(0 \le n \le 2^{N-1}\right)$, and p is a variable size list of additional parameters. The bits that are set in the base 2 representation of n mark the qubits that the gate acts upon. The effect of a quantum circuit on the quantum register is obtained by applying the gates in the order they appear in the GP individual.

The fitness of each individual is computed with respect to the error of the quantum computation. In (3), C is the set of fitness cases, r^c is the result of the quantum computation performed by the GP individual, and e^c is the expected result.

$$f = \sum_{c \in C} \left\| r^c - e^c \right\|^2 \tag{3}$$

The mutation operator works on a chromosome by choosing a gate at random and changing it with a new one. The crossover operator selects a random

cutting point within each parent and swaps the gates after the chosen points between the two parents. The selection scheme used is roulette wheel selection (also known as fitness proportionate selection) which constructs a new population from the old one, based on the fitness values of the individuals. Individuals with better fitness have a greater chance to survive in the new population. This selection scheme was enhanced by elitism – survival of the best individual is always guaranteed. Besides determining the quantum circuit that performs the required computation, the optimization process favors individuals that contain lesser elementary gates. We introduce an editing operator that removes unused gates (*i.e.* gates that are not meaningful to the result of the computation performed by the quantum circuit) on some randomly chosen individuals in each generation.

4.3. EXPERIMENTS

The genetic programming algorithm described in the previous section was applied to the problem of designing quantum circuits for the production of 2, 3, and 4 maximally entangled qubits in the form $\frac{1}{\sqrt{2}}\left(|0...0\rangle + |1...1\rangle\right)$. Hence, in each case there is one fitness case: the input register is prepared in the state $|0...0\rangle$ and the output register shoul be in the given entangled state. The set of quantum gates that the algorithm may choose from contains the gates: CNOT, NOT, Square NOT (SNOT), Hadamard (H), Simple Rotation (SR), Swap (Sw). The algorithm uses a population of 300 individuals, which evolve over 200 generations. The mutation operator rate is 1% and the crossover operator rate is 70%. The editing operator rate is 10%. For every problem we present the best solution obtained in 50 independent runs with afore mentioned parameters.

4.4. RESULTS

GP found perfect matching circuits for all three problem instances within the first 15 generations on the average. In runs when the editing operator rate is too low (*e.g.* $< 1\%$), these circuits display some form of redundancy, even though the algorithm favors parsimonious solutions. However, in this case, the editing operation may be applied to the final solution of the algorithm. The results presented in the paper were obtained in runs with a 10% rate of the editing operator, and we noted that these solutions presented no form of redundancy.

In each run, the GP algorithm managed to distinguish among the gates that are useful for the production of entangled states. Hence, although the function set contained various gates, on each run the solution found by the algorithm was a sequence of Hadamard and CNOT gates.

For 2 qubits systems, the algorithm always finds one of the two perfect solutions: H2·CN1[2], or H1·CN2[1], depicted in Fig. 2. For example, the solution H2·CN1[2] means that a Hadamard gate (H) is applied to the second qubit ($2_{(10)} = 10_{(2)}$), and then a Controlled Not gate (CN) is applied to the first qubit ($1_{(10)} = 01_{(2)}$) under the influence of the second qubit (the additional parameter 2), producing the quantum evolution from (4).

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \end{pmatrix} \tag{4}$$

For 3 qubits systems, the algorithm finds various perfect solutions: H1·CN2[1]·CN4[1], or H4·CN2[4]·CN1[4], or H2·CN4[2]·CN1[4], depicted in Fig. 3. For 4 qubits systems, some of the perfect solutions found by the algorithm are: H8·CN2[8]·CN4[2]·CN1[2], or H1·CN8[1]·CN2[8]·CN4[8], or H1·CN2[1]·CN4[1]·CN8[2].

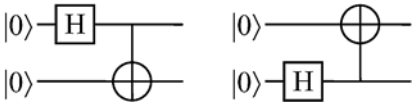

Fig. 2 – Quantum circuits for the production of 2 qubit entangled states.

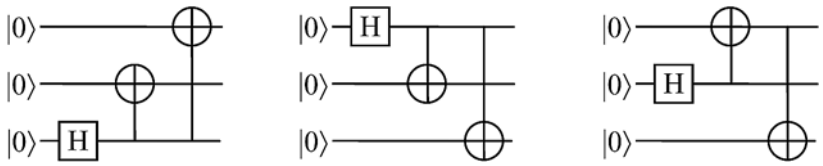

Fig. 3 – Quantum circuits for the production of 3 qubit entangled states.

5. CONCLUSIONS

Genetic programming is a promising methodology to automatically develop (new) quantum circuits and algorithms, a task that is rather difficult for human researchers. This paper presents a novel genetic programming scheme based on S-expressions adapted to the particular features of quantum circuits. The experimental results demonstrate the effectiveness and the applicability of this approach. Future work will include GP parameter tuning, searching of new algorithms, as well as finding optimized versions of known algorithms.

Acknowledgements. This paper was supported by the CNMP CEEX-05-D11-25/2005 Grant.

REFERENCES

1. P. Shor, *Algorithms for Quantum Computation: Discrete Logarithms and Factoring*, Proceedings of the 35th Annual Symposium on Foundations of Computer Science, 1994.
2. A. M. Turing, *Computing machinery and intelligence*, Mind, **59**, 433–460, 1950.
3. J. H. Holland, *Adaptation in Natural and Artificial Systems*, University of Michigan Press, 1975.
4. J. R. Koza, Genetic *Programming: On the Programming of Computers by Means of Natural Selection*, MIT Press, Cambridge, MA, 1992.
5. K. Han, J. Kim, *Quantum-inspired Evolutionary Algorithm for a Class of Combinatorial Optimization*, IEEE Trans. Evol. Comput. **6**, 580–593, (2002).
6. J. Jang, K. Han, J. Kim, *Quantum-Inspired Evolutionary Algorithm-Based Face Verification*, Lecture Notes in Computer Science (2724), Proc. Genetic Evolutionary Computation Conf. 2003, Springer-Verlag, 2147–2156, 2003.
7. K. Han, J. Kim, *Quantum-Inspired Evolutionary Algorithms with a New Termination Criterion, H€ Gate, and Two-Phase Scheme*, IEEE transactions on evolutionary computation, **8**(2), 2004.
8. C. P. Williams, A. Gray, *Automated Design of Quantum Circuits*. Lecture Notes in Computer Science (1509), Quantum Computing and Quantum Communications, Springer-Verlag, 1999.
9. F. Vatan, C. Williams, *Optimal Quantum Circuits for General Two-Qubit Gates*, Physical Review A (Atomic, Molecular, and Optical Physics), **69**(3), APS, 2004.
10. L. Spector, H. Barnum, H. J. Bernstein, N. Swamy, *Quantum Computing Applications of Genetic Programming*, Advances in Genetic Programming, **3**, 135–160, 1999.
11. Lee Spector, *Automatic Quantum Computer Programming: A Genetic Programming Approach*, Kluwer Academic Publishers, ISBN 1-4020-7894-3, Boston, 2004.