An Algorithm for Evolving Multiple Quantum Operators for Arbitrary Quantum Computational Problems

Walter O. Krawec
Stevens Institute of Technology
Hoboken NJ, 07030
walter.krawec@gmail.com

ABSTRACT

We design and analyze a real-coded genetic algorithm for the use in evolving collections of quantum unitary operators (not circuits) which act on pure or mixed states over arbitrary quantum systems while interacting with fixed, problem specific operators (e.g., oracle calls) and intermediate partial measurements. Our algorithm is general enough so as to allow its application to multiple, very different, areas of quantum computation research.

Categories and Subject Descriptors

I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search

Keywords

Quantum Computing; Quantum Algorithms; Real Coded Genetic Algorithm

1. ALGORITHM DESIGN

By now, several authors have investigated the use of evolutionary algorithms to construct quantum algorithms. The majority of this work (see for instance [4, 2]) consists of using evolutionary techniques to construct quantum circuits (collections of basic quantum gates). In [1], Hutsell and Greenwood first considered evolving, using an evolutionary strategy approach, not quantum circuits, but arbitrary quantum unitary operators. There are several advantages to both approaches: evolving quantum circuits seems more desirable for finding quantum algorithms; evolving arbitrary quantum operators though can be useful for more abstract work.

In this paper, we greatly extend the work of [1] by designing a real-coded GA that, given a rule set from the user which can be used to describe several very different problems, evolves multiple quantum operators which can act on arbitrary quantum systems over arbitrary pure or mixed initial states (the reader is referred to [3] for an introduction to quantum computation). More specifically, our algorithm evolves unitary operators $\{U_i\}_{i=1}^{\Lambda}$ conforming to a specified `RuleList` given by the grammar:

```
RuleList :- {(InitialState, Rule);}+
Rule :- {({Op}*) : (Pr)}+
Op :- U_i | V_i | M_i | M_{i,j} | T_i
Pr :- {p:[{space:basis}+]}+
InitialState :- (psi_0) | (pure, vector)
                | (mixed, matrix)
```

After specifying the dimension of the underlying Hilbert space $\mathcal{H}$ (and all underlying subspaces if any; that is, we construct $\mathcal{H} = \mathcal{H}_1 \otimes \cdots \otimes \mathcal{H}_n$), along with matrices describing the fixed, problem specific V_i operators (these may be oracle calls for example), the user next constructs a `RuleList`.

To compute the fitness of a candidate solution $\{U_i\}$ given a `RuleList`, each `Rule`, which is itself a list, is evaluated individually and each entry of this `Rule` is considered sequentially. To evaluate the i'th `Rule`, we first construct density matrix $\rho_{i,0}$ using the i'th `InitialState` description (which may be a matrix or vector description; alternatively it could be a default state $|\psi_0\rangle\langle\psi_0|$). After this, we apply all operators specified by the first `Op` list (these can include measurement operators M and mixtures of U, V, and M operators - these we call T operators) resulting in a new density matrix $\rho'_{i,0}$ from which we may compute the fitness with respect to the first `Pr` entry which is of the form:

$$\texttt{Pr}_{i,1} = p_1\texttt{:[space}_{1,1}\texttt{:basis}_{1,1}\texttt{,space}_{1,2}\texttt{:basis}_{1,2}\texttt{,}\cdots\texttt{]},\ p_2\texttt{:[space}_{2,1}\texttt{:basis}_{2,1}\texttt{,space}_{2,2}\texttt{:basis}_{2,2}\texttt{,}\cdots\texttt{]},\cdots$$

This list represents the rule that, with probability p_j, if subspace $\texttt{space}_{j,1}$ and $\texttt{space}_{j,2}$ etc. are measured, we should receive outcome $|\texttt{basis}_{j,1}\rangle$ and $|\texttt{basis}_{j,2}\rangle$ and so on. We assume the computational basis $|1\rangle, |2\rangle, \cdots$ for all measurements. Since we support V operators (which may be used to change bases), this is without loss of generality. Let $q_{i,j}$ be the actual probability of such a measurement outcome as specified by the j'th entry of $\texttt{Pr}_{i,1}$; this is computed as:

$$q_{i,j} := tr\left(\prod_k M(\texttt{space}_{j,k}, \texttt{basis}_{j,k})\rho'_{i,0}\right),$$

where $M(a,b)$ is the measurement operator, acting on subspace a projecting to $|b\rangle$; that is $M(a,b) = I \otimes |b\rangle\langle b|_a \otimes I$ where the first I is the identity on all subspaces to the left of $\mathcal{H}_a$ and the second is on all subspaces to the right. The fitness of the first entry of the `Rule` then is defined as: $f_{i,1} = \sum_{p_j \in \texttt{Pr}_{i,1}} (q_{i,j} - p_j)^2$.

GECCO'14, July 12–16, 2014, Vancouver, BC, Canada.
ACM 978-1-4503-2881-4/14/07.
http://dx.doi.org/10.1145/2598394.2598408.

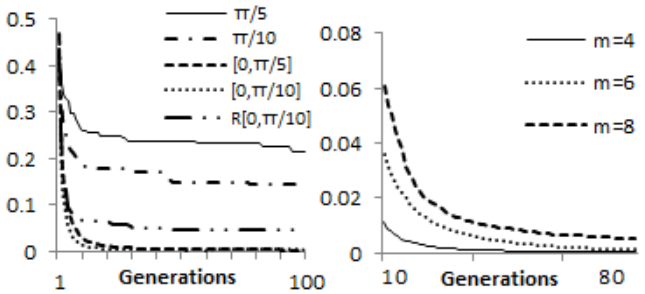

Figure 1: Left: Showing how step-size affects fitness of a population over each generation (using the Basic Experiment, with $m = 16$). The notation $[0, x]$ implies a single random value was chosen as the step size, in the specified interval, whenever mutate was first called; $R[0, x]$ implies a random value within $[0, x]$ was chosen individually for each value mutated. Right: Fitness of Deutsch-Jozsa test

To progress to the second entry of the i'th `Rule`, we repeat the above using $\rho_{i,1} = \sum_{j,k} M(j,k)\rho'_{i,0}M(j,k)$ as our new initial state, where the sum is over all subspaces j that were measured (determined by $\texttt{Pr}_{i,1}$) and all basis states k in that subspace. This density matrix represents the (mixed) state resulting from an unobserved measurement on these subspaces. After repeating for each entry of the i'th `Rule`, we compute the fitness of this rule, denoted f_i as: $f_i = \frac{1}{N_i}\sum_{j=1}^{N_i} f_{i,j}$ where N_i is the number of elements in the i'th `Rule`. After all `Rules` in the `RuleList` are evaluated in this fashion, the fitness of a candidate solution $\{U_i\}$ is taken to be the average of these: $f(\{U_i\}) = \frac{1}{N}\sum_j f_j$.

To represent a solution, we employ the same method as in [1], described in [5], which decomposes a $D \times D$ unitary matrix (with complex entries) into D^2 elementary unitary transformations and a single phase change, requiring D^2+1 real parameters in the range $[0, 2\pi]$.

Mutation will alter, for each operator U_i, 80% of these $D_i^2 + 1$ parameters (each U_i has its own parameters - furthermore, our algorithm allows operators to act only on subspaces of $\mathcal{H}$; thus each operator may have different dimension D_i) increasing or decreasing their values by an amount called the step size. We found setting this value to a random number in $[0, \pi/10]$ produced very good results. See Figure 1 (left). Crossover is simple one-point crossover, treating each U_i individually. From this we constructed a hybrid real coded GA (with elitism) with a localized hill-climbing search. After applying crossover (using tournament selection) and mutation, we take the very best solution and perform a local search on it (using the mutation routine).

We first evaluated our algorithm on the Basic Experiment described in [1] which involves the search for a single U_1 that, after applied to the state $|\psi_0\rangle = \frac{1}{\sqrt{m}}\sum_{i=1}^{m}|i\rangle$ in the m-dimensional space $\mathcal{H} = \mathcal{H}_1$, we should measure $|1\rangle$ with probability 1. Using our rule grammar this is simply:

```
((psi_0):(U.0):(1:[1:1]));
```

We ran this test for $m \in \{16, 20, 24, 32\}$. Our stopping condition was a fitness value of less than 0.01 which represents an incorrect measurement result (e.g., a measurement result of $|i\rangle_1$ for $i \neq 1$) of less than 10%. We note that the stopping condition used in [1] was to have an incorrect measurement result less than 30% of the time, an easier condition to achieve, and yet the GA/HC approach we use seems to converge remarkable faster. For example, with $m > 16$, they were, using a $(100 + 100)$ ES, unable to get convergence after more than 450 generations. We, however, for $m = 16$, achieved our stopping condition after less than 32 generations (average over 20 independent trial runs). For $m = 20$ we required less than 50, $m = 24$ required 60. Unfortunately we are not sure if this is strictly because of our GA+HC approach or perhaps we have fine-tuned our genetic parameters better (or some combination of the two). As Figure 1 (left) shows, this problem is highly susceptible to these parameters.

We also tested our approach by having it evolve an algorithm which functions similarly to the Deutsch-Jozsa algorithm [3] - a quantum algorithm which, after interacting with an oracle gate only once (the oracle implements a function $f : \{0, 1, \cdots, m-1\} \rightarrow \{0, 1\}$ which is either balanced or constant - these oracles are implemented as V operators in our algorithm), is able to determine a certain property of this oracle (namely, whether the function is balanced or constant). While solved using other evolutionary algorithms [4, 2], it serves as a good "stress-test" as it requires $\binom{m}{m/2}+2$ "V" operators and at least as many rules which are of the form:

```
((psi_0): (U.0, V.0, U.1):(1:[2:1]));
((psi_0): (U.0, V.1, U.1):(1:[2:1]));
((psi_0): (U.0, V.2, U.1):(0:[2:1]));
((psi_0): (U.0, V.3, U.1):(0:[2:1]));
...
```

Here V_0 and V_1 are the two constant oracles; the underlying Hilbert space is $\mathcal{H}_1 \otimes \mathcal{H}_2$ of dimension 2 and m respectively. See Figure 1 (right) for the results.

More information (including an extended version of this paper with further evaluations in other areas of quantum computation), and full source code for our implementation (written in C++) is available online: `walterkrawec.org/math/QuantumOp.html`.

2. REFERENCES

[1] S. R. Hutsell and G. W. Greenwood. Applying evolutionary techniques to quantum computing problems. In *IEEE Congress on Evolutionary Computation (CEC 2007)*, pages 4081–4085, September 2007.

[2] Paul Massey, JohnA. Clark, and Susan Stepney. Evolving quantum circuits and programs through genetic programming. In Kalyanmoy Deb, editor, *Genetic and Evolutionary Computation, GECCO 2004*, volume 3103 of *Lecture Notes in Computer Science*, pages 569–580. Springer Berlin Heidelberg, 2004.

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

[4] L. Spector. *Automatic Quantum Computer Programming: A Genetic Programming Approach*. Kluwer Academic Publishers, Boston, MA, 2004.

[5] Karol Zyczkowski and Marek Kus. Random unitary matrices. *Journal of Physics A: Mathematical and General*, 27(12):4235, 1994.