Meta-Genetic Programming For Static Quantum Circuits

Kenton M. Barnes
University of Warwick
Coventry, UK
k.barnes@warwick.ac.uk

Michael B. Gale
University of Warwick
Coventry, UK
m.gale@warwick.ac.uk

ABSTRACT

Quantum programs are difficult for humans to develop due to their complex semantics that are rooted in quantum physics. It is therefore preferable to write specifications and then use techniques such as genetic programming (GP) to generate quantum programs instead. We present a new genetic programming system for quantum circuits which can evolve solutions to the full-adder and quantum Fourier transform problems in fewer generations than previous work, despite using a general set of gates. This means that it is no longer required to have any previous knowledge of the solution and choose a specialised gate set based on it.

CCS CONCEPTS

• **Software and its engineering** → **Genetic programming**; • **Computer systems organization** → *Quantum computing*;

KEYWORDS

genetic programming, quantum computing

ACM Reference Format:
Kenton M. Barnes and Michael B. Gale. 2019. Meta-Genetic Programming For Static Quantum Circuits. In *Genetic and Evolutionary Computation Conference Companion (GECCO '19 Companion), July 13–17, 2019, Prague, Czech Republic.* ACM, New York, NY, USA, 4 pages.
https://doi.org/10.1145/3319619.3326907

1 INTRODUCTION

Quantum programs are programs that, to our best knowledge, can only be run in linear time complexity on a quantum computer. They take advantage of quantum mechanics to achieve exponential speed-ups compared to classical computers in some areas.

The semantics of quantum programs can be defined in terms of quantum mechanical principles such as superposition and entanglement. However, these principles are alien to us because they are unparalleled in our everyday lives. While this makes the programming of quantum computers a hard task for humans, the syntax of quantum programs remains surprisingly simple. For example, in the circuit model for quantum computation, programs are expressed as an arrangement of unitary transformations acting on quantum bits (qubits), called quantum circuits. A quantum program could, therefore, exist as a list of small data structures defining a unitary transformation, the qubits to act on, and any other parameters required for the transformation's construction. We refer to these small data structures *gates*. Such a representation is shown below:

```
H (Qubit 0)
UP (Qubit 0) (Angle pi/4)
CNOT (Qubit 0) (Qubit 1)
```

The unitary transformations for each of the gates used can be found in Appendix A. The qubits the gates act on are provided by an argument, and other parameters like the angle ϕ needed for the *UP* gate are also given. This syntactic simplicity suggests that some methods previously used to automatically generate programs for classical computers could be just as applicable in the domain of generating quantum circuits.

One such method is *genetic programming* (GP), which is the use of genetic algorithms to manipulate and create programs. Systems that use GP are largely unaware of the semantic meaning of the program elements they manipulate. Instead, for a given input, they calculate a fitness value by comparing a candidate program's output with the desired output.

These GP systems can be augmented using *meta-genetic programming*, which is where some parts of the genetic algorithm itself are able to evolve and change alongside the candidate solutions. The aim of this is to remove the need for a human to optimise the genetic algorithm for each problem they want to solve, and instead have the genetic algorithm optimise its own methods. The way this is achieved varies from having the whole genetic algorithm written in a language that can be genetically manipulated, or just having a few parameters that are optimised during the running of the genetic algorithm.

In this paper, we continue efforts to use GP for producing quantum programs. Concretely, our contribution is a meta-genetic programming system which combines the following techniques to allow us to evolve quantum circuits without prior knowledge about a particular problem's solution:

- To determine how likely it is that a particular type of quantum gate is added to a candidate solution during mutation, we assign probabilities to each type of gate. Our system allows for these probabilities to also be mutated on the meta-level when evolving a problem solution.
- We present a general gate set comprised of quantum gates and *higher-order gates* that can be used to evolve quantum programs successfully, without the need for the manual optimisation of this gate set.
- We compare our system to previous work and show that, for a given problem, it can evolve a solution in fewer generations even whilst using a general set of gates rather than a specialised one.

GECCO '19 Companion, July 13–17, 2019, Prague, Czech Republic

ACM ISBN 978-1-4503-6748-6/19/07…$15.00
https://doi.org/10.1145/3319619.3326907

2 BACKGROUND AND PREVIOUS WORK

Koza [1] pioneered the use of genetic algorithms for automatic program generation for classical computers. Spector et al. [7] then applied this method to quantum programs and set out several techniques for representing populations of quantum programs for use in GP systems. The technique that this paper focuses on is referred to as the *static* method, which generates abstract representations of quantum circuits. The qubits that the circuits act on are first set to an input state, the circuit is then run, and then the resultant states of the qubits are measured. As these quantum circuits have a fixed number of qubits, the number of qubits required to represent the input could be larger than the number of qubits available. Thus, this method is not scalable, and there will always be some size of input that a static circuit cannot deal with.

A common weakness of previous work using this representation is that the collection of gates (the *gate set*) that the quantum circuits are to be made up of is manually selected. GP system designers often use information about an optimal solution for the problem they wish to solve in order to inform the selection of their gate set. For example Massey et al. consistently improve the gate set used by their system by hand in order to achieve better results [2–4]. In particular they choose to add gates they know helps them evolve a circuit that implements the quantum Fourier transform (QFT).

While a chosen gate set may represent some circuits very succinctly, it may also make others very complicated. It is likely to not be very useful for evolving solutions to problems other than the ones it is optimised for. A natural solution to trying to create a more "generic" gate set that can be used to solve many problems is to make it larger, which is more likely to be able to succinctly represent more circuits. However, this has the cost of slowing evolution down as there are more gates that can be added to a circuit during mutation, and therefore it is less likely that the optimal gate would be placed. In this paper, we are addressing this issue through the use of meta-genetic programming.

3 METHOD

In our system, we focus on improving the static approach of evolving abstract representations of quantum programs. The resulting programs can then be scaled up by hand, which we believe is significantly less difficult than developing a solution without assistance from an automatic programming system in the first place. Specifically, we use *stackless linear genetic programming* [7], which creates circuits of the form seen in the Introduction. The complete list of gates, along with their unitary transformations, defined for use in our language can be found in Appendix A. We use *tournament selection* to pick candidate solutions (*individuals*) to form the parents of the next generation: to select a single individual using this method we select, at random, a specific number of individuals, called the *tournament*. We then select the fittest circuit in this tournament to be included in the pool of parents for the next generation with a given probability. If the fittest was not selected then we select the next fittest with the same probability, until an individual is chosen. The system also has *elitism*: the automatic preservation of a given percentage of the fittest individuals. Our system uses the fitness function, genetic mutation, and genetic crossover methods described by Spector [6]. Genetic mutation was influenced by Potoček et al. [5]: it can use 3 of their proposed 12 genetic operators. Initialisation of circuits is *ramped*: a maximum size is defined and then the size of the randomly produced circuits is distributed uniformly between 1 and the maximum size.

3.1 Probabilities of adding gates

When mutating candidate solutions, new genetic material is often added. In our case, this comes in the form of new gates that are added to the circuit. The gate that is chosen comes from the gate set which is specified as an input to the genetic algorithm. While in previous work this gate set is an optimised collection of gates for solving the problem at hand, our work removes the requirement to optimise this gate set. Instead, we give the system a comparatively large gate set and expect it to optimise this down to the gates that are found to be useful during evolution.

Each individual has, in addition to its quantum circuit representation, a probability distribution of gates. When a candidate solution is undergoing a mutation that requires new genetic material, the gate is chosen probabilistically according to that individual's gate distribution. At the start of evolution the distributions are uniform: each gate has an equal chance of being produced during a mutation.

During evolution, the distributions can undergo genetic crossover or mutation. These operations are applied with a given probability and are independent of the mutation and crossover of the quantum circuits. The crossover operation will produce a distribution that is the average of the two given distributions. The mutation operation can decrease or increase the probability of a specific gate being produced. There are two possible methods for the selection of the gate to change the probability of, which we explore: it could be uniformly at random or the chance of increasing the probability a given gate occurs could be proportional to the amount of times the gate appears in the circuit of the individual. In either case, mutation works such that all gates are never completely removed from the distribution, but instead can just have an arbitrarily small probability of occurrence. This feature is in place to allow gates to "come back" and occur more commonly again if they become more relevant in future generations, though the ability of the genetic algorithm to do this is limited as it takes many generations for this to occur. However, the minimum probability that all gates can be assigned is configurable: setting a larger value for this minimum makes it easier for gates to come back later in the process, but also reduces the effectiveness of the distribution as it reduces the ability for it to remove useless gates from the system.

3.2 Higher-order gates

It is difficult for the designer of a GP system to know what gates should be included in the gate set. It is even more difficult to create a gate set that could be used to solve a wide array of problems. Most designers go with one of two options to choose their gate set. Many will create a gate set that has "generic" gates that are applicable to a wide range of problems. This has the disadvantage that there are often gates which will be better for the specific problem at hand, compared to these commonly used gates. On the other hand, many designers solely choose gates that are seen in the human-designed algorithms that solve the problem at hand. While this can mean the system is able to solve the problem quickly as it has

Table 1: Results of evolving full-adder and quantum Fourier transform solutions.

	Population	Gate set	Probability Manipulation	Successful[1]	Mean Gens.
Full-adder	100	General	None	0%	N/A
	100	General + Higher	None	98.2%	179 ± 8.7
	100	General + Higher	Random	97.8%	170 ± 7.3
	100	General + Higher	Frequency Proportionate	93.0%	210 ± 17.4
	200	Massey	Frequency Proportionate	99.2%	93 ± 6.5
	200	Massey	N/A	N/A	943[2]
Quantum Fourier	1000	General + Higher	None	99.8%	231 ± 13.9
	1000	General + Higher	Random	99.6%	208 ± 12.3
	1000	Potoček	None	91	108 ± 7.4
	1000	Potoček	N/A	98%	1053[3]

the best gates available to do so, this process requires knowledge about the solution to the problem. This may not be possible if we want to apply GP to previously unsolved problems or want to use the same system for a range of problems. Another disadvantage of specialised gate sets is that it is often the case, and often the aim, that GP produces solutions that solve problems in ways that humans have not thought of. By limiting the system to the specialist gates that the designer feels should be used to solve the problem we reduce the ability of the system to solve the problem in new ways.

A compromise between these two approaches is through the use of *higher-order gates*. These are functions that take other gates as arguments, to produce more "complicated" gates. The intention is to have a gate set consisting of several higher-order gates as well as some "generic" gates, such that the genetic algorithm can now produce, via the higher-order gates, a large array of more complicated gates. This gives the genetic algorithm access to complicated gates that may be the best to solve the problem in a way that does not require knowledge about the problem at hand. This approach works well with the meta-genetic approach of changing the probabilities that different gates are added. This is because, if a specific higher-order gate and generic gate pair produce a useful output, then both these gates will become more likely to be added to a circuit during a mutation. It is important to optimise the gate set in this way to make sure mutations are using the more relevant and useful gates. Otherwise many mutations would be wasted, as the gate set for this approach is quite large and likely to contain many gates that are redundant for the current problem.

4 RESULTS

To evaluate our system, we used it to evolve solutions for the full-adder (FA) and quantum Fourier transform (QFT) problems using different combinations of gate sets. Each experiment was repeated 500 times. We recorded the mean number of generations that was required to arrive at a solution. The results of our experiments are shown in Table 1 while the values used for the system's parameters

[1]The process was stopped after 500 generations for the fuller-adder problem and 3000 for the quantum Fourier problem. If the fitness value was not within 0.1% of the optimal fitness by that point, the attempt was deemed unsuccessful.
[2]This value is quoted from the work of Massey et al. [4].
[3]This value is quoted from the work of Potoček et al. [5].

Table 2: Hyper-parameters

Variable	FA	QFT
Mutation Rate	0.8	0.8
Crossover Rate	0.6	0.6
Elitism	0.05	0.05
Tournament Size	5	8
Max. Initial Circuit Size	10	20
Meta-Mutation Rate	0.4	0.025
Meta-Crossover Rate	0.1	0.01
Meta-Occurrence Rate	0.025	0.05

are shown in Table 2. The gate sets that we use are defined as:

$$\begin{aligned}\text{General} &= \{CNOT, PX, PY, PZ, UP, H, SWAP\}\\ \text{Higher} &= \{Conditional, Anti\text{-}conditional, All, Inverse\}\\ \text{Massey} &= \{PX, RX, RZ, RY, V, W, CNOT, CCNOT\}\\ \text{Potoček} &= \{H, UP, CUP, CCUP, SWAP\}\end{aligned}$$

The *Massey* and *Potoček* sets are inspired by the works of Massey et al. [4] and Potoček et al. [5] respectively, and are included to compare our system to their work.

The *General* set contains common gates seen in a large variety of quantum circuits. It is not specialised and we intend to use this gate set for solving arbitrary problems. The higher-order gates introduced in Section 3.2 are grouped into *Higher*. We use them to enhance the *General* set as described in that section.

5 DISCUSSION

We selected the FA and QFT problems in order to be able to compare the results we obtained with our system to those obtained in previous work. However, the novel methods discussed in this paper are not the only variants when comparing results to previous work: genetic operators, hyper-parameters, and specifics of the SLGP representation all slightly differ. The results shown in Table 1 indicate that our system is able to evolve solutions in fewer generations than Massey et al. [4] and Potoček et al. [5] for the FA and QFT problems whilst using the same gate sets as them. We hypothesise that this is due to better-tuned hyper-parameters and a more effective genetic mutation operation. It continues to be able

to surpass previous results with our *General + Higher* gate set, and hence without prior knowledge of the problem solutions.

The full-addition problem can be solved quickly if an optimal gate set is used. Using *e.g.* {*CCNOT*, *CNOT*}, the optimal solution requires only four gates. However, without knowledge of the solution we might not have the more obscure *CCNOT* (Toffoli) gate in our gate set. Instead, we might use a more standard gate set such as the *General* set, which does not include the Toffoli gate. In this case, the optimal solution consists of around 30 gates which is larger and therefore harder for a genetic algorithm to evolve. This can be seen in the first result in Table 1, where the experiment that used the *Generic* gate set failed to solve the problem in 500 generations.

Adding higher-order gates to the general gate set allows us to successfully evolve the adder in 179 generations, only 1.9x slower than using the optimised gate set. This can be attributed to the *Conditional* higher-order gate, as it can create the equivalent of the *CCNOT* gate while being more general. No manual gate set optimisation or knowledge about a problem's solution is required. We improve on this further with the meta-genetic method (Section 3.1) by an average of 9 generations using the random method. When using the frequency proportionate method, the number of generations increased, and thus the random method was used for the remaining experiments.

The results for evolving QFT are similar. Using the *General + Higher* gate set, we evolved solutions in roughly $\frac{1}{4}$ as many generations as Potoček et al. [5] and only 2.1x slower than using their optimised gate set in our system. When using the right hyper-parameters for probability mutation and crossover rates, enabling the meta-genetic feature slightly decreased the mean generations to 208. This is only a factor of 1.9x off the same system being used with a manually optimised gate set.

6 CONCLUSIONS AND FURTHER WORK

We presented a quantum programming system which can evolve solutions to the FA and QFT problems in fewer generations than previous work, but without needing prior knowledge about a solution. If given prior knowledge in the form of a manually optimised gate set, our system is even more efficient.

Our main improvement is the addition of higher-order gates which increase the effectiveness of a general gate set. Enhancing this approach by dynamically evolving the probabilities that certain types of gates are chosen in the mutation stage led to further, but small, improvements. The combination of these two techniques allows our system to reach solutions with this generalised approach only about 2x slower than using a manually optimised gate set.

Our system also performs better than previous work when using exactly the same gate sets that they used. For example, to solve the QFT problem our system required 9.8x fewer generations than the work of Potoček et al. [5]. We theorise that this improvement is because of our more effective genetic mutation operation and the use of well optimised hyper-parameters.

Our work paves the way to more general, genetic algorithms that can evolve solutions to quantum programming problems which do not depend on human guidance or previous knowledge of the solution. However, there is more room for improvement. Throughout this work the system user has had to optimise hyper-parameters such as the population size and mutation rate manually. More work can be done on methods to tune these automatically. Also, as shown in the results, our system is still more efficient with an optimal gate set. Ideally, there would be no difference in the use of an optimal gate set versus a general gate set. We envisage that further improvements in this area could be achieved by exploring the addition of new higher-order gates.

A GATE DEFINITIONS

$$CNOT = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \quad SWAP = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad CUP = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e^{i\phi} \end{bmatrix}$$

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \quad UP = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\phi} \end{bmatrix} \quad V = \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix} \quad W = \begin{bmatrix} 1 & 0 \\ 0 & -i \end{bmatrix}$$

$$PX = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \quad PY = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \quad PZ = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} \quad RZ = \begin{bmatrix} e^{-i\theta} & 0 \\ 0 & e^{i\theta} \end{bmatrix}$$

$$RX = \begin{bmatrix} cos\ \theta & -i\ sin\ \theta \\ -i\ sin\ \theta & cos\ \theta \end{bmatrix} \quad RY = \begin{bmatrix} cos\ \theta & -sin\ \theta \\ sin\ \theta & cos\ \theta \end{bmatrix}$$

$$CCUP = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & e^{i\phi} \end{bmatrix} \quad CCNOT = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

B HIGHER-ORDER GATES

- *Conditional*: a given operation is performed if a specified qubit is 1.
- *Anti-Conditional*: a given operation is performed if a specified qubit is 0.
- *Inverse*: the inverse of a given operation is performed.
- *All*: a given operation is applied to all possible qubits.

ACKNOWLEDGMENTS

We would like to thank the referees for their constructive feedback and useful suggestions about the paper.

REFERENCES

[1] John R Koza. 1990. *Genetic programming: A paradigm for genetically breeding populations of computer programs to solve problems.* Vol. 34. Stanford University, Department of Computer Science Stanford, CA.

[2] Paul Massey, John A Clark, and Susan Stepney. [n. d.]. Evolution of a human-competitive quantum fourier transform algorithm using genetic programming. In *Proceedings of the Genetic and Evolutionary Computation Conferece 2005 (GECCO '05).*

[3] Paul Massey, John A Clark, and Susan Stepney. 2004. Evolving quantum circuits and programs through genetic programming. In *Proceedings of the Genetic and Evolutionary Computation Conferece 2004 (GECCO '04).* Springer, 569–580.

[4] Paul Massey, John A Clark, and Susan Stepney. 2006. Human-competitive evolution of quantum computing artefacts by genetic programming. *Evolutionary Computation* 14, 1 (2006), 21–40.

[5] Václav Potoček, Alan P Reynolds, Alessandro Fedrizzi, and David W Corne. 2018. Multi-objective evolutionary algorithms for quantum circuit discovery. *arXiv preprint arXiv:1812.04458* (2018).

[6] Lee Spector. 2004. *Automatic Quantum Computer Programming: a genetic programming approach.* Vol. 7. Springer Science & Business Media.

[7] Lee Spector, Howard Barnum, Herbert J Bernstein, and Nikhil Swamy. 1999. Quantum computing applications of genetic programming. *Advances in genetic programming* 3 (1999), 135–160.