Implementing a Quantum Circuit Optimizer Using Genetic Programming and ZX-Calculus

Rachel Huang
College of William & Mary
Williamsburg, VA, USA
rhuang03@wm.edu

Pieter Peers
College of William & Mary
Williamsburg, VA, USA
ppeers@wm.edu

***Abstract*—Quantum computers are implemented on networks of subatomic particles and are thus extremely fragile, causing circuit error to grow exponentially as their complexity increases. ZX-calculus is an optimization technique that applies a set of simplifying equivalence rules to circuit operations, removing unnecessary components and reducing their size. This project implements a quantum circuit optimizer that generates target circuits and optimizes them for ZX-calculus using genetic programming. Generated circuits are evaluated for their accuracy, optimization time, length, and suitability for ZX-calculus. We validate the genetic programming experiments on the Torino QPU simulator and generated circuits are evaluated on the Kingston QPU simulator provided by the IBM Quantum Platform.**

***Index Terms*—Quantum Computing, ZX-calculus, Genetic Programming**

I. Introduction

Quantum computing is a growing field of computer science that utilizes the quantum mechanical properties of subatomic particles to solve complex problems. Traditional binary bits are replaced by qubits, quantum particles that exist in a superposition of 0 and 1. The value of a qubit is only determined after measuring the qubit, which physically collapses it to one of the states based on the probabilities defined by its superposition [Schneider and Smalley, 2023a]. In addition, qubits can be induced into entanglement, a phenomenon in which individual states become dependent on the states of other qubits in the system [Schneider and Smalley, 2023b]. Due to these properties, quantum circuits display a clear advantage over classic circuits when applied to problems consisting of dynamic, multivariable interactions.

Current work in quantum computing is usually oriented towards one of the following directions: creating quantum implementations of existing classical algorithms or discovering new algorithms in which quantum properties would provide a significant advantage. An example of a quantum approach to a classical problem is Shor's algorithm, which provides an exponential speedup for integer factorization so that the task only takes polynomial time. On the other hand, [Neven and Smelyanskiy, 2025]'s report on Google's Quantum Echoes algorithm demonstrates a novel algorithm that achieves unprecedented precision with its usage of qubit constructive interference properties. However, the current physical reality of quantum circuits is not able to match the theoretical capabilities of quantum computing. Quantum circuits are extremely fragile and incur large amounts of error proportional to the complexity of a problem, thus motivating the development of optimization techniques that could potentially reduce the size of quantum circuits and decrease the qubit area on which error could accumulate [Schneider and Smalley, 2023a].

The ZX-calculus optimization technique rearranges and simplifies circuit operations to remove unnecessary components. ZX-calculus is composed of two parts: the ZX-diagramming language and a set of ZX-rewrite rules. To apply ZX-calculus to a circuit, the circuits are first converted into ZX-form where components are represented as two different kinds of angle-based operations. The optimizer then iterates through the fixed set of ZX-rewrite rules until a desired threshold of optimization is reached [van de Wetering, 2020].

In this paper, we apply genetic programming to randomly initialized quantum circuits with the goal of producing a set that satisfies a target functionality and that is optimized for the most effective application of ZX-calculus. This project closely aligns with NASA's Science Mission Directorate, as NASA is studying the applications of quantum technology to its current endeavors. For example, [Stamper-Kurn et al., 2023]'s whitepaper details how NASA is developing space-based quantum networks for secure information exchange. NASA's current quantum technology relies on photon states, whose extreme fragility have been the primary limitation of quantum computation. Generating circuits that are primed for the application of ZX-calculus optimization has the potential to decrease the number of photons necessary to implement desired computations. This could potentially improve circuit stability and consequently the accuracy, speed, and computing power of existing quantum technology. To this end, we opt for genetic programming as an evolutionary machine learning strategy because of its powerful ability to search a given problem space with only minimal starting information [Rubinstein, 2001]. Since our experiments begin with a randomly initialized population, genetic programming was the ideal choice for generating meaningful subsequent circuits with the goal of priming circuits for ZX-calculus optimization. Genetic programming mutates and prunes an initial population to find a set that optimizes the selected fitness criterion. The fitness function determines the shape of the search space by filtering which individuals and features are allowed to continue to the next generation; in our case, the fitness function prefers

individuals that are shorter in length and more similar to the target in accuracy and structure.

Three target circuits from the IBM Qiskit Python SDK are chosen for our experiments: the CDKMRippleCarryAdder, the RGQFTMultiplier, and the WeightedAdder. These circuits are part of the Qiskit circuit library and perform various arithmetic operations on qubit registers. This paper implements a custom fitness function that considers the states of the generated and target circuits, its accuracy on the chosen problem set, and the effectiveness of ZX-calculus when applied to the circuit. We generate a set of circuits for each target and evaluate them on their problem spaces, showing that their circuit states match the target states exactly, but their functional accuracy for their problem space can still be significantly improved. We believe that our genetic programming framework has the potential to generate functionally accurate quantum circuits if the experiment parameters are strategically modified, and that our ZX-calculus evaluation scores will have an increased impact as the complexity of the selected targets is also increased.

II. Related Work

We focus our discussion of related work to prior research that applies genetic programming to the task of generating quantum circuits.

A. Fitness Functions

Genetic programming experiments use a fitness function to determine which individuals continue to the next generation. Since the fitness function determines the shape of the search space, the criteria components guide the genetic algorithm to optimize the population for specific characteristics. [Stein and Farber, 2024] construct their fitness functions so that circuits are optimized for maximum quantum advantage, with quantum advantage being defined as the amount of entanglement and superposition present in an individual. To do so, the fitness function enforces constraints that prefer circuits with more entanglement-inducing gates, such as the CNOT gate. In this manner, the genetic algorithm is incentivized to first favor circuits that had a greater quantum advantage before improving performance on the target problem set. Other fitness functions may prioritize the generated circuit state's similarity to the target state. To this end, [Rubinstein, 2001] sum the differences in the amplitudes between the generated and target states, and [Creevey et al., 2023] square the inner product of the amplitudes between the respective states. Summing the state differences prefers a smaller fitness value, while squaring prefers larger fitness values. Our method implements a custom fitness function that focuses on improving generated state similarity to the target, improving the effectiveness of ZX-calculus, and minimizing generated circuit length.

B. Genetic Operators

Genetic operators allow the genetic algorithm to move through the search space by mutating individuals and introducing new features into each generation. Crossover and mutation operators are used, where crossover operators determine how features from two individual circuits contribute to producing a child circuit for the next generation. [Rubinstein, 2001] define crossover as an uniform or unweighted operation that could be applied to each level of an individual's structure; uniform crossover selected components with an equal chance and weighted crossover selected components with respect to their associated weights. [Creevey et al., 2023]'s simple crossover operation combined alternating halves of the two parent circuits to produce a child circuit. For mutation operators, [Brady et al., 2014] implement a set that included swap, insertion, replacement, and deletion operations that used a basis set from which all individuals were built. Our experiments use uniform crossover and a relatively high mutation rate of 50% to introduce variety into the population since our basis set is small.

C. Applications

Targets for genetic programming experiments can vary in complexity. Simple targets will likely produce solutions that match the target state exactly; for example, [Rubinstein, 2001] generate circuits that yielded a fully accurate solution for the EPR maximal entanglement circuit of various sizes. On the other hand, [Stein and Farber, 2024] chose the Bernstein-Vazirani problem and Unstructured Database search problem as their targets, which were both relatively more complex search problems. Thus, the generated circuits do not yield exact target matches and are were evaluated with regard to their average fitness values for each generation. [Creevey et al., 2023] generated W-state entanglement circuits that were able to have a lower gate count and lower complexity than the default circuits used by Qiskit. Our chosen targets are Qiskit arithmetic circuits and also relatively simple, as our priority is to add a novel component to the fitness function that optimizes individuals for ZX-calculus.

III. Method

Each experiment follows the same genetic programming pipeline:

1) Initialize the circuit population Section III-A
2) Evaluate the fitness of the individuals in the population (Section III-B).
3) Check if the maximum iterations or desired fitness threshold parameters have been reached
4) Breed the population using the crossover operator (Section III-E)
5) Mutate the population based on the randomly selected mutation operator and the mutation rate parameter (Section III-E)
6) Optimize the angle values of each individual's gates with consideration of the individual's disposition to ZX-calculus (Section III-C,(Section III-D).
7) Select the individuals that continue to the next generation based on their fitness value and the survival rate parameter. Repeat steps 2-7 until either of the stopping conditions in step 3 is met.

Algorithm 1 Fitness function incorporating aspects of [Creevey et al., 2023, Rubinstein, 2001, Stein and Farber, 2024].

$test\_cases \leftarrow sample\_problem\_space()$
$avg\_svec\_val = 0$
$avg\_testcase\_val = 0$
for $test\ in\ test\_cases$ **do**
    $avg\_svec\_val\ +=\ calculate\_fitness(circuit, test)$
    $avg\_testcase\_val\ +=\ evaluate\_test(circuit, test)$
end for
$avg\_svec\_val\ /= len(test\_cases)$
$avg\_testcase\_val\ /= len(test\_cases)$
return $avg\_svec\_val + avg\_testcase\_val$

Algorithm 2 ZX-calculus evaluation function; ZX-calculus implementation based on the PyZX $full\_reduce$ method [Madison, 2025].

$circuit \leftarrow bind\_angle\_parameters(circuit)$
$circuit, pre\_zx, post\_zx \leftarrow apply\_zx\_calc(circuit)$
$pre\_stats = parse\_circuit\_stats(pre\_zx)$
$post\_stats = parse\_circuit\_stats(post\_zx)$
$differences = [\ ]$
for $gate\_type\ in\ stats$ **do**
    $diff = pre\_stats[key] - post\_stats[key]$
    if $pre\_stats[key]\ != 0$ **then**
        $diff\ /= pre\_zx\_stats[key]$
    end if
    $differences\ +=\ \frac{diff}{100}$
end for
return $\frac{sum(differences)}{len(differences}$

Algorithm 3 Angle optimization algorithm used to optimize each gate in a circuit.

$fitness\_cost = fitness\_cost\_function(angles)$
$zx\_cost = zx\_cost\_function(angles)$
if $zx\_cost\ != 0$ **then**
    $zx\_cost = \frac{1}{zx\_cost}$
end if
$length\_cost = \frac{1}{len(circuit)}$
return $(w_1 \cdot fitness\_cost) + (w_2 \cdot zx\_cost) + (w_3 \cdot length\_cost)$

A. Initialization

We use the Qiskit $random\_circuit$ method to create a randomly initialized population of individuals for the first generation of each experiment. Our basis gate set, $\{R_X, R_Y, R_Z, CNOT\}$, provides an input to $random\_circuit$ so that each of the individuals will be composed of the same gates. In addition, the $circuit\_depth$ parameter of $random\_circuit$ is set to a constant value of 7, so that the randomly initialized individuals will all be of relatively equal complexity. The $R_i\ \forall\ i\ \in\ [X, Y, Z]$ operations take a rotation angle as input and apply it around the i axis, while the CNOT gate is a 2-qubit operation that flips the target qubit if the control qubit is in the 1 state. The operations in our basis gate set can be categorized into two types: 1-qubit gates $\{R_X, R_Y, R_Z\}$ and 2-qubit gates $\{CNOT\}$.

B. Fitness Function

We evaluate the fitness of the circuit individuals with our proposed custom fitness function. The fitness function samples the problem space and iterates through each test case. Since our selected targets are Qiskit arithmetic circuits, the problem space consists of input and circuit pairs, where the input is a tuple of operands, and the circuit implements the operands and the appropriate target arithmetic circuit. For each test case, the Qiskit Statevector similarity and the differences in the arithmetic operation results are recorded. The fitness value of an individual is thus represented by the sum of these two comparison values, as shown in Algorithm 1.

C. ZX-calculus Evaluation

The angle optimization step of each individual's gates factors in the degree to which a circuit is primed for ZX-

calculus. This is quantified by parsing the gate statistics of a circuit before and after applying ZX-calculus. Each parameterized circuit individual is bound to its angle values and the percentage difference in the number of each basis gate is recorded. The average of the differences thus represents how effective the application of ZX-calculus is for the circuit individual, as shown in Algorithm 2

D. Angle Optimization

The fitness and ZX-calculus evaluation values returned by Algorithm 1 and Algorithm 2 are used to optimize the angle parameters of each gate during the angle optimization step of the genetic programming pipeline. The normalized fitness, ZX-calculus, and length values for each circuit are assigned a weight $w_i \; \forall \; i \in [1, 3]$ corresponding to the emphasis we placed on each attribute. $w_i = 0$ means that we exclude the attribute i in our analysis of the circuit's optimization value. The angle optimization cost function thus returns the weighted sum of the circuit's fitness, ZX-calculus, and length components that we are interested in, as shown in Algorithm 3. [SciPy Community, 2008]'s $minimize$ method from the $optimize$ module minimizes the cost function with regard to the angle values.

E. Crossover and Mutation Operators

Table I lists the crossover and mutation operators implemented to address steps 3 and 4 of the aforementioned genetic programming pipeline. All mutation operators have an equal probability of being selected during the mutation step in a generation. The crossover and mutation operators introduce a variety of features into the population as described in Table I, allowing the genetic algorithm to traverse the circuit space.

IV. Experiments

We perform a series of experiments using the genetic programming framework described in the previous sections. The purpose of these experiments is to test our genetic pipeline for generating quantum circuits that are optimized for ZX-calculus. We are also interested in developing an evaluation function for quantifying how effective ZX-calculus is for a circuit. Our selected targets are the Adder (CDKMRippleCarryAdder) target that computes the sum of two qubit registers in-place, the Multiplier (RGQFTMultiplier) that computes the product of two qubit registers out-of-place, and the WAdder (WeightedAdder) that computes the weighted sum of its input qubit register [CDK, 2023, RGQ, 2023, WAd, 2023]. We perform 6 experiments total for the 3 selected targets, with 1 experiment per target incorporating the consideration of ZX-calculus into its angle optimizations, and 1 experiment per target not incorporating the consideration of ZX-calculus into its angle optimizations. Our experimental goal is to generate a set of circuits per experiment that satisfies the target arithmetic functionality and that is optimized for the most effective application of ZX-calculus. The complete set of experiments is detailed in Table II, with the first entry in the Fitness Function Weights column corresponding to w_1, the second entry corresponding to w_2, and the third entry corresponding to w_3.

Table II lists the genetic programming parameters used for the experiments. A decimal value indicates the percentage of the population to which the corresponding operation is applied. Different parameters are chosen for the experiments to account for the difference in complexity between the chosen targets and the subsequent amount of time it takes for genetic operations to complete.

We execute each experiment on a Qiskit AerSimulator instance modeled after the IBM Torino Quantum Processing Unit (QPU). The IBM Torino QPU consists of 133 qubits, with $\{CZ, ID, R_X, R_Y, R_Z, R_{ZZ}, SX, X\}$ as its basis gate set. Each circuit is transpiled to the hardware basis for execution on the simulator, and the Layout stage of the Qiskit transpilation pipeline maps the virtual qubits to the hardware qubits.

V. Results

We execute circuit analyses on a Qiskit AerSimulator instance modeled after the IBM Kingston QPU.

A. WAdder (WeightedAdder) Experiments

A sample of 100 circuits is randomly selected from each of the WAdder experiments' final populations. From these samples, Experiment 1 returned 94 non-empty circuits and Experiment 2 returned 85 non-empty circuits. Figure 1 shows the development of the average fitness, length, ZX-value, and optimization time metrics of the genetic algorithm for Experiments 1 and 2.

The experiments progress through a different number of generations due to external operations varying in time; executing on the IBM QPU simulator takes variable amounts of time depending on the status of the backend instance. Experiments 1 and 2 show similar convergence behavior for the custom fitness value of the population as defined by Algorithm 1. Analysis of the circuit samples demonstrates that the Statevector similarity of the circuits converges to 1, and since the Algorithm 1 custom fitness value is defined to be the sum of the Statevector similarity and the test case accuracy, the test case accuracy converges to approximately 5% for both experiments. The negative ZX-values can be explained by the ZX-calculus algorithm decomposing operations into the ZX-calculus language and our ZX-value being defined as a circuit's size reduction before and after applying ZX-calculus. Since our circuits are relatively simple, the increase in circuit size from ZX-decomposition is not offset by the size reduction effect the ZX-calculus algorithm has on complex circuits. The length of the circuit populations also decreases over time for both experiments.

Figure 2 lists the best performing circuits that were generated during Experiment 1. In this context, the Statevector Similarity circuit has the highest similarity to the target circuit as determined by Qiskit's Statevector class. The Target Test Case Accuracy circuit has the highest accuracy on the test cases sampled from the WAdder problem space, and the Algorithm 1 Fitness Value circuit has the highest sum of these

Operator	Effect
Crossover	Given a crossover-index i and two parent circuits c_1, c_2, the gates from c_2 with index $> i$ are appended to the gates from c_1 with index $< i$ [Creevey et al., 2023].
Swap Mutation	Given two gates g_1, g_2 in a circuit, g_1 is replaced with g_2 and g_2 is replaced with g_1 [Brady et al., 2014].
Insert Mutation	Given a gate g_1, g_1 is inserted into the circuit at location occupied by gate g_2. Increases the length of the circuit individual [Brady et al., 2014].
Replace Mutation	Given a gate g_1, g_1 is replaced with another gate of the same type [Creevey et al., 2023].
Delete Mutation	Given a gate g_1, g_1 is removed from the circuit individual. Decreases the length of the circuit individual [Brady et al., 2014].

Table I: *Implementation details for selected operators*

ID	Target	Fitness Function Weights	N Qubits	Circuit Depth
1	Adder	0.5, 0.3, 0.2	8	7
2	Adder	0.7, 0.0, 0.3	8	7
3	Multiplier	0.5, 0.3, 0.2	8	7
4	Multiplier	0.7, 0.0, 0.3	8	7
5	WAdder	0.5, 0.3, 0.2	11	7
6	WAdder	0.7, 0.0, 0.3	11	7

Table II *Executed experiments*

ID	Initial Population Size	Number of Qubits	Circuit Depth	Mutation Rate	Survival Rate	Desired Fitness	Max Iterations	Minimum Population Size
1	100	8	7	0.5	0.65	3.0	100	20
2	100	8	7	0.5	0.65	3.0	100	20
3	100	8	7	0.5	0.65	3.0	100	20
4	100	8	7	0.5	0.65	3.0	100	20
5	1000	11	7	0.5	0.65	3.0	100	200
6	1000	11	7	0.5	0.65	3.0	100	200

Table III *Experiment parameters.*

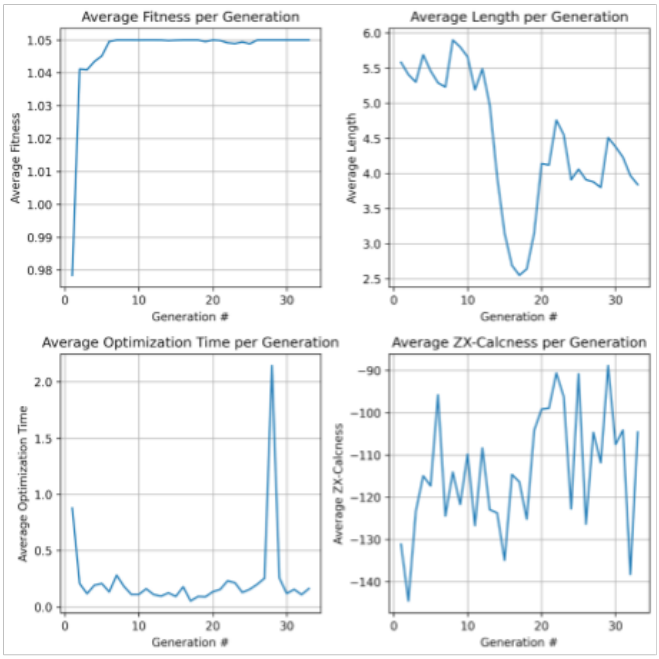
two values, which is what we used to filter circuits in our experiments. Figure 3 lists the best performing circuits generated by Experiment 2 corresponding to the same aforementioned attributes.

In the case of the Statevector Similarity circuits from Experiments 1 and 2, the circuit's Statevector matches the Statevector of the Qiskit WAdder circuit exactly: [1.+0.j 0.+0.j 0.+0.j ... 0.+0.j 0.+0.j 0.+0.j]. For experiment 1, the best performing circuit as defined by Algorithm 1 achieves a 12.5% accuracy on the full problem space, with a ZX-value of -550 as calculated by Algorithm 2. For experiment 2, the best performing circuit also achieves a 12.5% accuracy on the full problem space, with a ZX-value of -550. These ZX-values correspond to a 5.5x increase in the number of gate operations for each circuit, due to the translation of operations into the ZX-language and the simplicity of the original circuits.

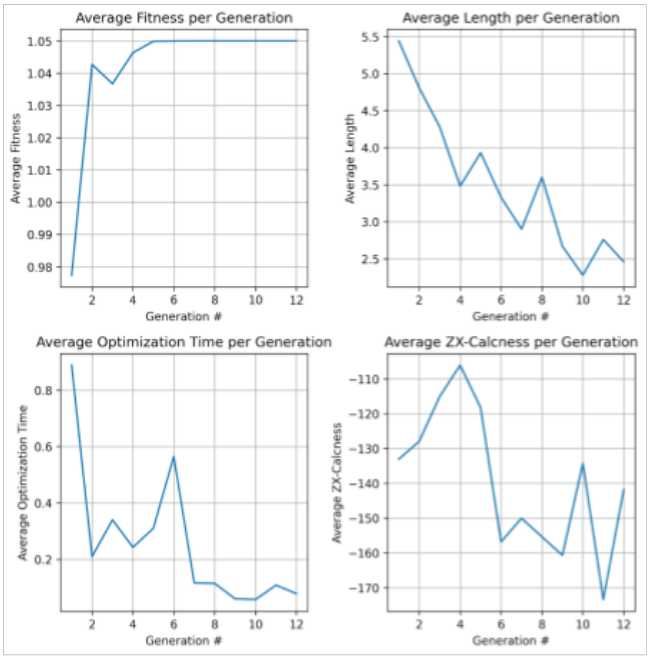
B. Multiplier (RGQFTMultiplier) Experiments

Experiment 3 returned 20 non-empty circuits and Experiment 4 returned 20 empty circuits. Figure 4 shows the development of the average fitness, length, ZX-value, and optimization time metrics of the genetic algorithm for Experiments 3 and 4. A minimum population size parameter is set, but due to a programming oversight, there was no lower bound set on circuit size. Algorithm 3 prefers shorter circuits, and in combination with the randomness introduced by step 4 of the genetic algorithm, the size of the individuals drops off to 0. The custom fitness value as defined in Algorithm 3 converges to approximately 1.05 for circuits in Experiment 3, with their Statevector Similarity converging to 1.0. Thus, the test case accuracy converges to approximately 5%. The length of the circuits in Experiment 3 also decrease in length over time.

Figure 5 lists the best performing circuits that were generated during Experiment 3. The best performing Statevector Similarity circuit from Experiment 3 matches the Statevector of the Qiskit RGQFTMultiplier circuit exactly: [1.+0.j, 0.+0.j, 0.+0.j, ..., 0.+0.j, 0.+0.j, 0.+0.j]. For experiment 3, the best performing circuit achieves a 23.4% accuracy on the full problem space, with a ZX-value of 42.0289855. The ZX-value corresponds to an approximate 42% decrease in the number of gate operations for the circuit. The best performing circuit as defined by Algorithm 1 has a Statevector Similarity value of 0.999999815, with a Statevector of [0.18464375+0.98280542j,0.+0.j,-0.00007923-0.0004217j , ...,



(a) Recorded metrics for Experiment 1



(b) Recorded metrics for Experiment 2

Fig. 1: Recorded metrics for WAdder Experiments. Experiment 1 considered a circuit's ZX-value in its optimization of the circuit's angles, while Experiment 2 did not.

0.+0.j,0.+0.j,0.+0.j].

C. Adder (CDKMRippleCarryAdder) Experiments

Experiments 5 and 6 returned empty circuits, due to the lack of a lower bound on the circuit size for individuals. Figure 6 shows the development of the average fitness, length, ZX-value, and optimization time metrics of the genetic algorithm for Experiments 5 and 6. The custom fitness value converges to less than 1 in both experiments, and the length of the circuits converges to 0.

VI. Discussion

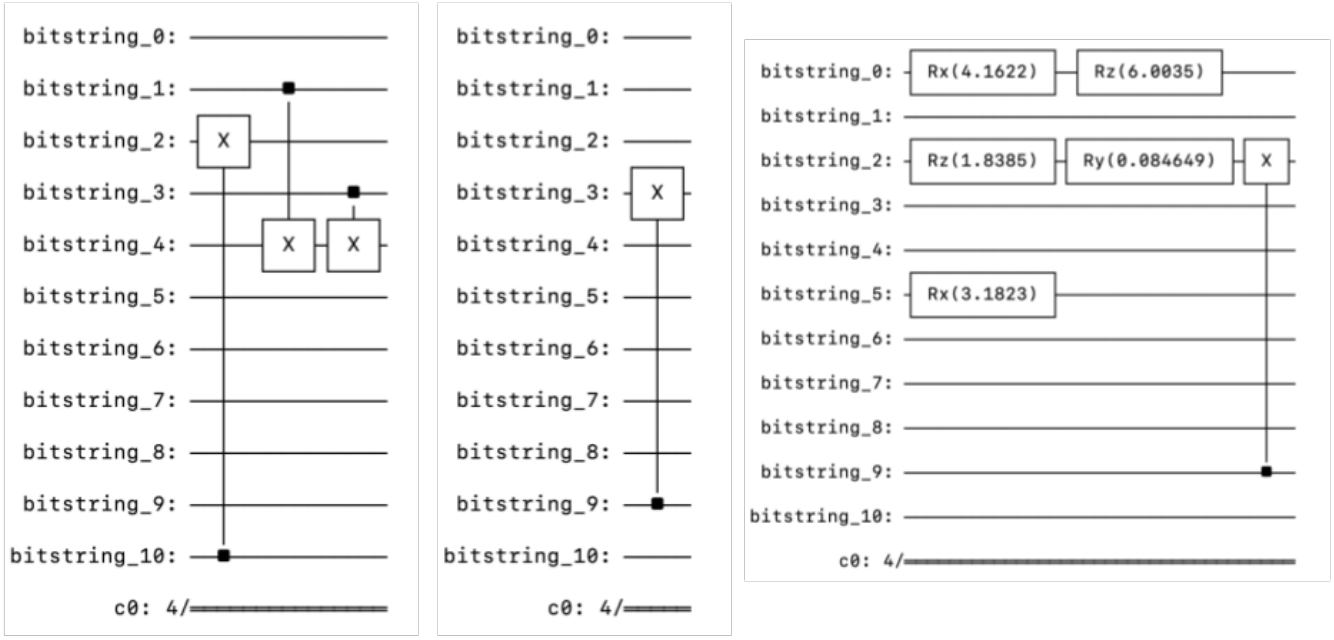
Our primary contribution, the proposed genetic search framework, generates quantum circuits that are able to match the target circuit states exactly, as quantified by the Qiskit Statevector class. However, this match in circuit states does not necessarily correspond to a match in the functionality between the generated circuits and the target circuits. Specifically, the generated quantum circuits are not able to achieve a high accuracy on the target problem spaces. Nonetheless, the general trends of the metrics we tracked for each experiment show that the genetic search framework has the potential to increase the target test case accuracy if the experiment parameters are modified and the run time for each experiment is increased. If time permitted, each experiment should also be ran for multiple iterations to average out the results for each target. Differences in experiment output can be attributed to the element of randomness introduced by Section III-E.

The large negative values for the ZX-evaluation scores of the WeightedAdder circuits demonstrate how the WeightedAdder circuit is too simple for ZX-calculus to significantly reduce the size of the optimized circuit. However, the increase in circuit size due to circuit decomposition into the ZX-language tends to decrease the optimization time for each circuit and increase its accuracy on the target problem space. For the relatively more complex RGQFTMultiplier target, ZX-calculus was able to reduce the size of the generated circuits in Experiment 3.

The empty circuit samples returned by Experiments 4, 5, and 6 are due to a programming oversight where there was no lower bound placed on the circuit size. Section III-D prefers shorter circuits, and in combination with this programming oversight and the element of randomness, the population for these experiments tends towards empty circuits.

VII. Conclusion

We introduced a genetic search framework for generating quantum circuits that are optimized for the application of ZX-calculus and satisfy a target functionality. As part of



(a) Statevector Similarity (b) Algorithm 1 Fitness Value (c) Target Test Case Accuracy

Fig. 2: Best performing circuits from WAdder Experiment 1

the framework, we proposed Algorithm 1, Algorithm 2, and Algorithm 3 to filter circuits based on their state similarity to the target, their accuracy on the target problem space, and the effectiveness of ZX-calculus on the circuit. We demonstrated the effectiveness of our method by describing 6 experiments that generate circuits for the chosen targets of the Qiskit CDKMRippleCarryAdder, RGQFTMultiplier, and WeightedAdder arithmetic circuits.

Our search strategy equally favors a circuit's state similarity to the target and a circuit's accuracy on the target problem space, with an additional consideration for the circuit's disposition to ZX-calculus. For future research we would like to fine-tune our experiment parameters and our fitness function weights, in order to ensure that our genetic programming framework is best tailored for emphasizing our chosen characteristics. Other avenues for research include expanding the basis gate set for our genetic programming framework and selecting more complex targets to test the robustness of our framework and see if more complex gates are more appropriate for increasing the disposition of generated circuits to our chosen characteristics. We believe that our genetic programming framework has the potential to generate functionally accurate quantum circuits that are primed for the application of ZX-calculus, and can be applied to other quantum circuit generation tasks that would benefit from having a diverse and pre-optimized set of circuits.

ACKNOWLEDGMENT

This research was supported by a Virginia Space Grant Consortium Undergraduate Research Scholarship.

REFERENCES

Cdkmripplecarryadder. https://quantum.cloud.ibm.com/docs/en/api/qiskit/qiskit.circuit.library.CDKMRippleCarryAdder, 2023.

Rgqftmultiplier. https://quantum.cloud.ibm.com/docs/en/api/qiskit/qiskit.circuit.library.RGQFTMultiplier, 2023.

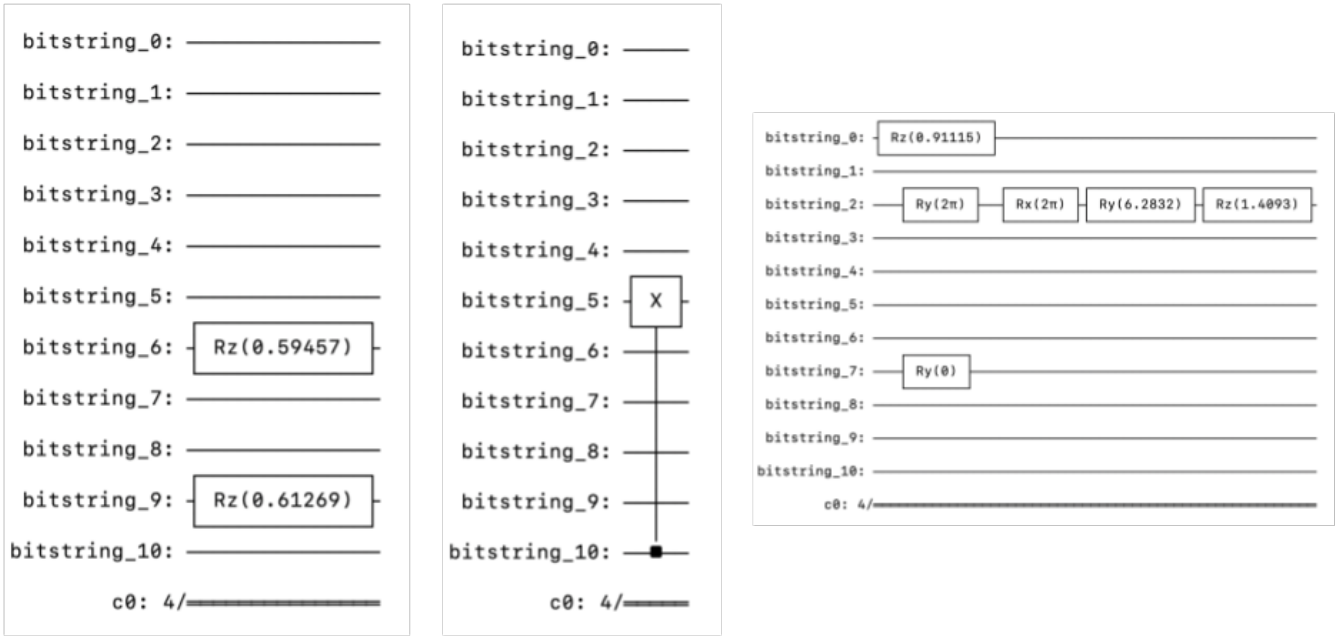
Weightedadder. https://quantum.cloud.ibm.com/docs/en/api/qiskit/qiskit.circuit.library.WeightedAdder, 2023.

A. Brady, J. Lawrence, P. Peers, and W. Weimer. genbrdf: Discovering new analytic brdfs with genetic programming. *ACM Transactions on Graphics*, 33(4):114, 2014.

F. Creevey, C. Hill, and L. Hollenberg. Gasp: a genetic algorithm for state preparation on quantum computer. *Scientific Reports*, 13:11956, 2023.

Lara Madison. Simplify. https://pyzx.readthedocs.io/en/latest/notebooks/simplify.html#Optimizing-circuits-using-the-ZX-calculus, 2025.

Hartmut Neven and Vadim Smelyanskiy. Our quantum echoes algorithm is a big step toward



(a) Statevector Similarity (b) Algorithm 1 Fitness Value (c) Target Test Case Accuracy

Fig. 3: Best performing circuits from WAdder Experiment 2

real-world applications for quantum computing. https://blog.google/innovation-and-ai/technology/research/quantum-echoes-willow-verifiable-quantum-advantage/, 2025. October 2025.

B. I. P. Rubinstein. Evolving quantum circuits using genetic programming. In *Proceedings of the Congress on Evolutionary Computation*, 2001.

J. Schneider and I. Smalley. What is quantum computing? https://www.ibm.com/think/topics/quantum-computing, 2023a.

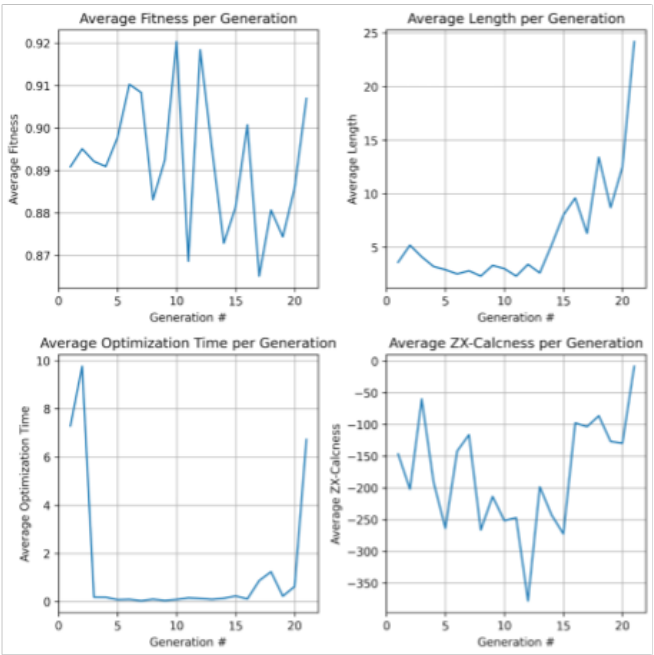
J. Schneider and I. Smalley. What is a qubit? https://www.ibm.com/think/topics/qubit, 2023b.

SciPy Community. scipy.optimize.minimize. https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.minimize.html, 2008.

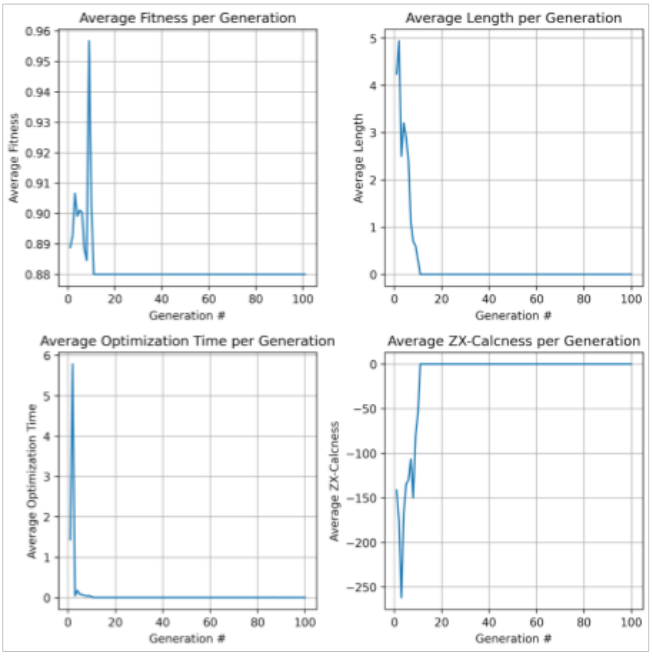
Dan Stamper-Kurn, Joshua Isaacs, Subhayan Roy Moulik, Sara Mouradian, and Kunal Marwaha. Topical: Quantum technologies in space. https://science.nasa.gov/wp-content/uploads/2023/05/194_d48f1fb875c016f97021c15addadb1ab_Stamper_KurnDanM.pdf, 2023.

Christoph Stein and Michael Farber. Incorporating quantum advantage in quantum circuit generation through genetic programming. *ACM Transactions on Quantum Computing*, 37(4):111, 2024.

J. van de Wetering. Zx-calculus for the working quantum computer scientist. *arXiv preprint arXiv:2012.13966*, 2020.

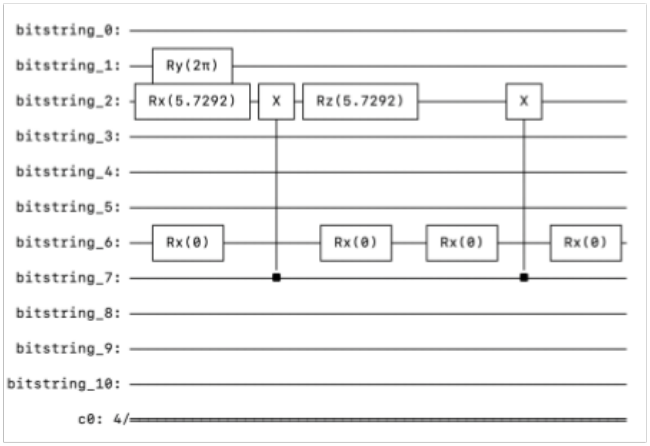


(a) Recorded metrics for Experiment 3

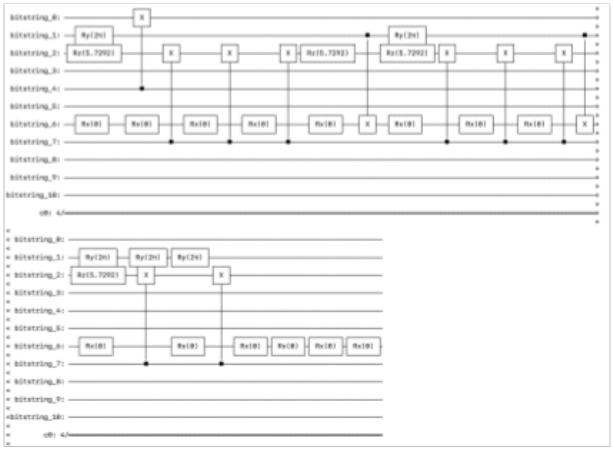


(b) Recorded metrics for Experiment 4

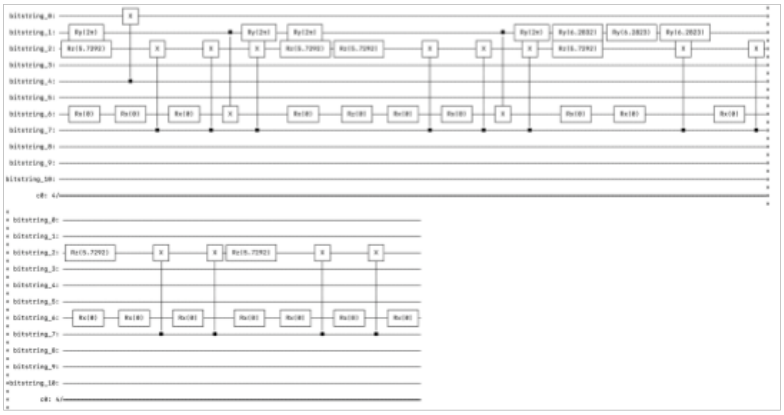
Fig. 4: Recorded metrics for Multiplier Experiments. Experiment 3 considered a circuit's ZX-value in its optimization of the circuit's angles, while Experiment 4 did not.



(a) Statevector Similarity

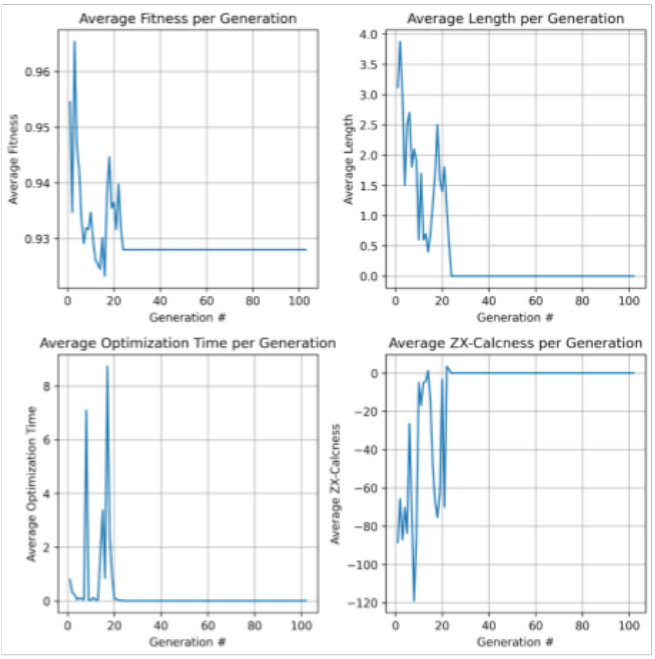


(b) Algorithm 1 Fitness Value

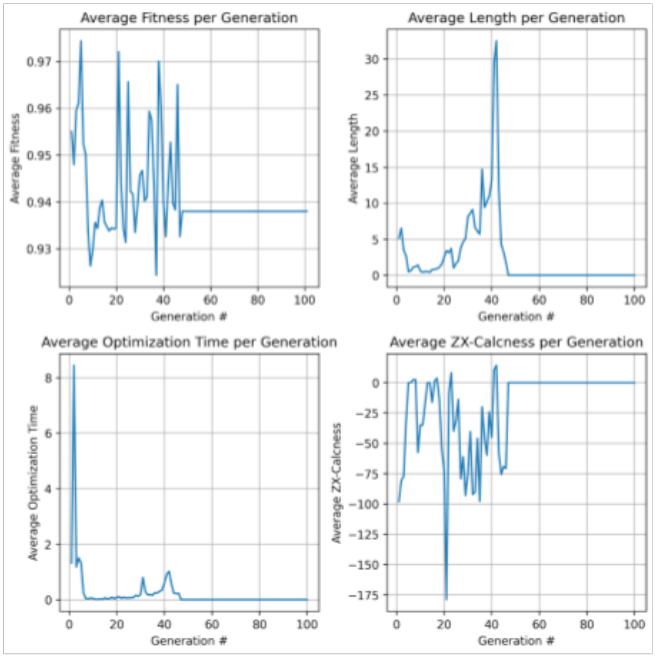


(c) Target Test Case Accuracy

Fig. 5: Best performing circuits from Multiplier Experiment 3



(a) Recorded metrics for Experiment 5



(b) Recorded metrics for Experiment 6

Fig. 6: Recorded metrics for Adder Experiments. Experiment 5 considered a circuit's ZX-value in its optimization of the circuit's angles, while Experiment 6 did not.