Applying Evolutionary Algorithms to Automatically Design Quantum-Gate Circuits

Benjamin Theron

April, 2024

Abstract. As we emerge into a new era of computing, the quantum age, discovering new ways to automate the design of the quantum circuits these systems rely on has become a subject of great interest. Being more research-oriented, this paper applies an evolutionary algorithm with a range of parameters and operators to examine its ability to generate quantum circuits of various sizes for two different quantum algorithms. The design of this algorithm, its development process and the methods used to test it are described. Additionally, benchmark results for the quantum algorithms in question are systematically compared against the results of the program. Experiments which alter the methods and parameters used by the evolutionary algorithm are executed and their impacts examined. An evaluation and holistic critical assessment about the success of the project as a whole is provided. Finally, this paper outlines what can be done to extend both the evolutionary algorithm and the project.

I certify that all material in this dissertation which is not my own work has been identified.

Contents

1 Introduction

This section of the paper introduces the motivation behind this project, providing the fundamental knowledge required to understand the problem at hand and the two quantum algorithms examined by the project.

1.1 Motivation

As we have rapidly progressed through the information age, humanity now finds itself at the precipice of the quantum age. With transistors continually decreasing in size, some now even smaller than a Nanometer [1], and with classical computers seemingly approaching their theoretical limits [2], quantum computers have been heralded as the solution that will take us into the future, excelling at tasks such as prime number factorisation [3] and quantum mechanical simulations [4]. Due to this, the importance of developing quantum applications and quantum computers has exploded over the last few decades.

The innate complexity of designing quantum algorithms means that their size and density when scaled up to a larger size, as would be required by more practical applications, quickly becomes intractable, permitting even experts from manually designing them with great success. This is amplified by the fact that quantum computers are specialised systems that operate on the smallest possible scale - often modelled on singular atomic particles - and can easily be interfered with via a wide range of external "noise", including heat, sound waves, electromagnetic waves, etc. [5]. So in spite of recent developments and a rise in the number of quantum experts, designing the circuits for quantum algorithms remains a arduous and error prone task. Additionally, quantum computers are currently very expensive and are likely to stay that way for a while, hence, any algorithm which could automatically design quantum circuits and thereby lessen the resources required to run a quantum system, would be of interest.

1.2 Background Knowledge

Classical computers operate on and store bits, where these are single units of information represented by either a zero or a one. By contrast, quantum computers utilise *qubits*, which are units of information that can be a zero, one or exist in a *superposition* of those two states (where it is both zero and one simultaneously). Schrödinger's cat is a mainstream thought experiment that illustrates the superposition of a particle [6]. Qubits are pieces of quantum information and represent the state of a quantum particle. Until the state of this particle is measured it exists in the aforementioned superposition of states, where this is the probability amplitudes of the particle existing in either state. The most common quantum systems off which qubits are modelled from are the spin of an electron or the polarisation of a photon.

Mathematically, these systems are represented using *bra-ket* notation, where *kets* are column vectors in a complex vector space which physically represent the quantum state of a particle - in this case the spin or polarisation in a given plane of measurement. The bra-ket notation for the spin of an electron would be $|0\rangle$ to represent spin down and $|1\rangle$ to represent spin up. Orthogonal vectors are vectors moving perpendicular to each other. An *orthogonal basis* is therefore two unit kets - a ket with a length of one - that are orthogonal to one another. The mathematical model for measuring the vertical spin of an electron can then be represented by the following orthogonal basis:

$|\uparrow\rangle = \begin{bmatrix}1\\0\end{bmatrix}$, $|\downarrow\rangle = \begin{bmatrix}0\\1\end{bmatrix}$, where this is known as the *standard basis*.

Measuring a qubit will collapse it from its superposition into one of its possible states, in the case of the electron you find that the particle is either spinning vertically upward or vertically downward. To operate on qubits, quantum algorithms must employ a series of quantum gates, which apply transformations to a given qubit(s) without measuring them. A quantum algorithm is therefore, the application of a range of quantum gates to create a desired output given some input qubit(s). A *quantum circuit* is a low-level representation of a quantum algorithm containing the collection of quantum gates, wires and functions - sometimes referred to as oracles - it requires. This is all in direct contrast to classical algorithms which, when provided with a string of binary inputs, apply a series of logical gates to produce a single binary output. Quantum circuits have the uniquely interesting property of being universally *reversible*, they essentially come with a re-do button that allows, with absolute certainty, for the inputs of a quantum algorithm to be found when provided with the outputs. This also means that quantum gates never lose information as qubits remain entangled before during and after passing through a quantum gate.

Optimising a quantum circuit typically involves reducing the number of gates and/or using more efficient gates (finding a modified gate set). Evolutionary algorithms (EAs) have proved effective at optimising quantum circuits against these objectives. The subset of EAs that have proved the most successful are *genetic algorithms* [7], which

are EAs that use a genetic representation, showing exceptional results when applied to quantum [8] and analogue circuits [9].

Evolutionary algorithms are a set of stochastic optimisation algorithms which use principles found in biological evolution. A population of candidate solutions each with randomly created traits are generated and have their fitness value - a measure of how much a solution varies from the target solution - calculated. A series of *operators* are then applied to this population, creating a new generation containing a - hopefully - fitter population. This process is then repeated until the generation limit is reached or the target solution is found. The genetic operators typically applied to the population include:

- Mutation: At a give rate, a single gene in an individual is randomly altered.
- Crossover: At a given rate, two individuals swap a subset of their genes.

Other optimisation algorithms such as swarm algorithms provide useful benchmark results [10] to compare against but contained research too recent and too lacking to be considered over genetic algorithms.

2 Project Specification

This segment explores the requirements of the project (both functional and otherwise), as well as its success criteria.

2.1 Grover's Search Algorithm

This algorithm finds, with near certainty, a desired state being looked for in an unordered database. A function known as an *oracle* is queried by the algorithm and confirms that the value being looked at is the correct value. Moreover, this algorithm has been shown to be asymptotically optimal [11], running in $O(\sqrt{N})$, which is quadratically faster than its classical counterpart, which runs in $O(N)$. The steps of the algorithm are as follows:

1. Invert the probability amplitude associated with the quantum state we are trying to locate.
2. Amplify the target probability amplitude. To do this, *Grover's Diffusion Operator* is used. This operator finds the mean probability for each of the qubits and flips each qubits probability amplitude about this mean [1]. This is labelled as the *diffusion transform* in Figure 1.
3. Apply Grover's operator the optimal number of times[2] [12].
4. Measure the resulting quantum state.

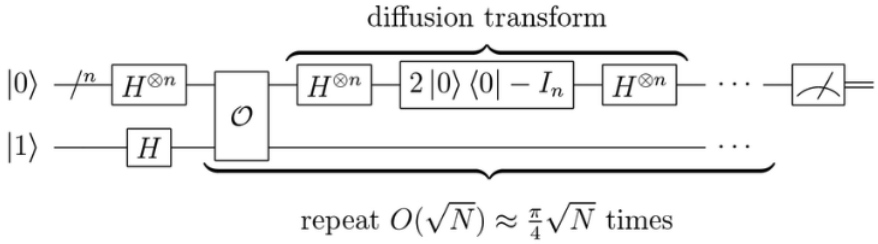


Figure 1: A n qubit example circuit design for Grover's algorithm [13].

2.2 The Quantum Fourier Transform

Acting on the quantum state vector of a qubit, this algorithm essentially transforms the quantum state from the computational basis into the Fourier basis. This algorithm works by applying a series of Hadamard gates and controlled phase shift gates, where these place an input qubit into a superposition of states and shift the phase (the rotation of the quantum particle the qubit is modelling) of an input qubit respectively. This process is applied to each of the input qubits and the resulting state is measured.

[1] For example, if the mean value is 0.5 and the target qubit has a probability of -0.2, this value is flipped to 0.7.

[2] $\frac{\pi}{4\arcsin(m/2^q)}$, where m is the number of marked states and q is the number of qubits.

The quantum Fourier transform (QFT) has been shown to require exponentially fewer operations than its classical counter part [14], requiring $O(N^2)$ gates as opposed to $O(n2^n)$, therefore providing an exponential speed up over its classical variant. The application of the QFT on n qubits is represented by the below equation [15]:

$$|j\rangle \mapsto \frac{1}{2^{n/2}} \sum_{k=0}^{2^n-1} e^{2\pi i j k/2^n} |k\rangle \tag{1}$$

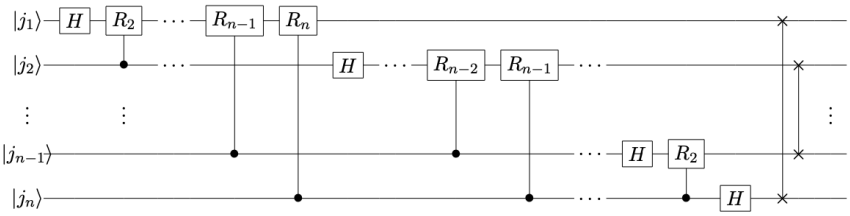


Figure 2: An n qubit example circuit design for the Quantum Fourier Transform [16].

2.3 Functional Requirements

The functional requirements state the processes that the program must incorporate in order to be successful, these include the following:

2.2.1 Main executable function: Executes the entire EA, executing the initialisation procedure, looping through the genetic algorithm for the specified number of iterations, and calling the data plotting function.

2.2.2 Initialisation procedure: Generates and stores a randomly created initial population of valid circuits.

2.2.3 Selection procedure: A subset of a provided population can be selected via tournament selection.

2.2.4 Mutation procedure: At a given mutation rate, a random gene in a provided candidate solution will be changed to another member in the provided gate set.

2.2.5 Crossover procedure: At a given crossover rate, two individuals will be selected and have a randomly selected subset of their genes swapped.

2.2.6 Circuit conversion function: Converts a circuit stored in the proprietary representation used by this project into a Qiskit object that can execute Qiskit methods.

2.2.7 Fitness evaluation function: A value indicating how far a given solution deviates from the target solution can be calculated.

2.2.8 Quantum Algorithms: Formulates circuits and gate sets for both the QFT and Grover's search for different qubit values.

2.2.9 Data logging and plotting procedure: Key data from runs of the EA are stored and able to be displayed as a visual diagram at the end of a run.

2.4 Non-Functional Requirements

Non-functional requirements outline supplementary procedures that enhance the capabilities of a project but are not required for it to be successful. For this project this includes:

2.3.1 The program files are well commented and comprehensible: The project files are well structured, well commented, with sensible function and variable names, thereby minimising the amount of time it takes an external party to understand the code.

2.3.2 The source code and project files are accessible: All files related to the project are stored in a Jupyter Notebook and are kept on a public GitHub repository to reduce the effort taken to reproduce the project.

2.3.3 The project executes completely and is efficient: To meet the standards required by Fujitsu Research and the ECM3401 module, the program must be free of bugs and should produce an output in a sensible time frame.

2.3.4 The project must be versatile: The functions and source code must have a high degree of modularity, being able to be altered in a plug and play manner that allows for easy experimentation and reproducibility.

3 Design and Implementation

This part of the paper outlines the design documentation and implementation of the main EA, along with any auxiliary functions and files.

3.1 Technical Architecture

The below diagram covers the structure of the source code, showing the key attributes and functions within each of the files that constitute the program. Additionally, the specific classes, methods and attributes used from the Deap library are included to present a full picture of the program structure.

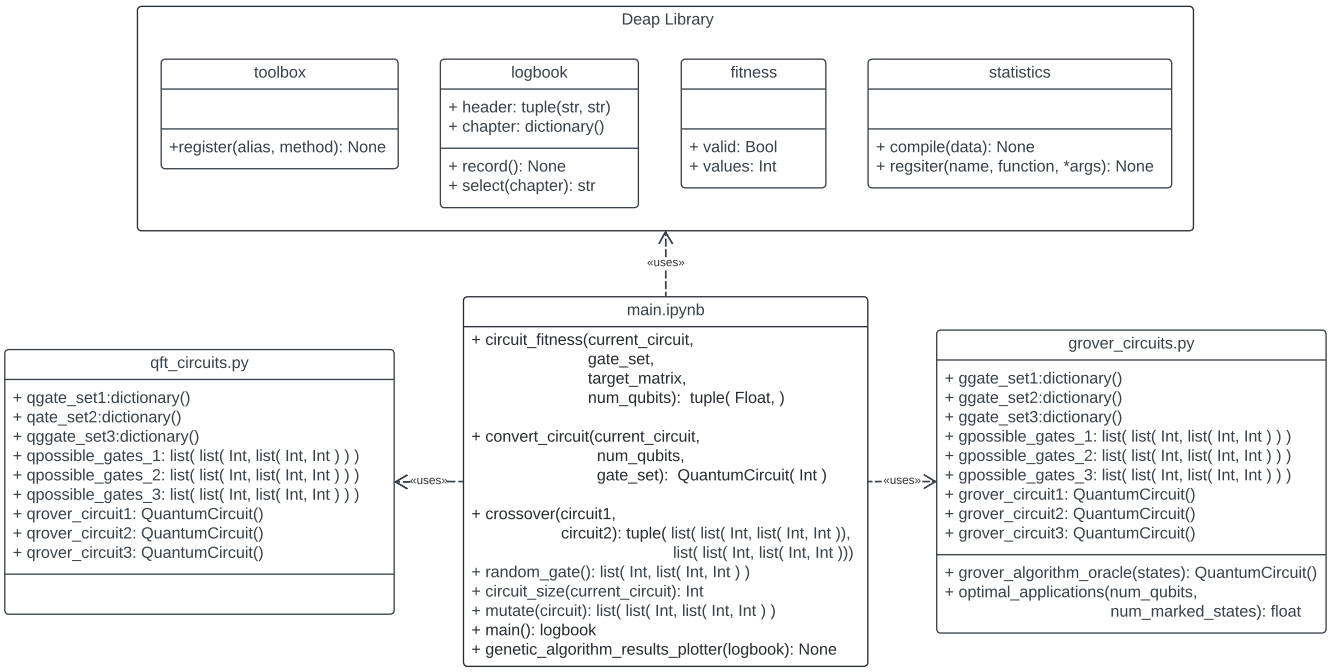
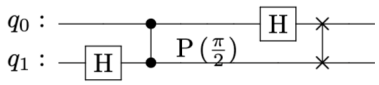


Figure 3: A UML Function Diagram Showing the Structure and Functionality of the Program Files.

3.2 Quantum Circuit Representation

The representation schema used to store quantum circuits is one of the defining characteristics of any system that uses these circuits and directly impacts both the time and space complexity of any algorithm that the system uses.



circuit = [[1, [1]], [3, [0, 1], [10, [1], [1, [0]], [2, [1, 0]]

gate_set = {1:HGate(), 2:SwapGate(), 3:CPhaseGate(math.pi/2), 10:"WIRE"}

Figure 4: The Gate Set and Representation Used to Store the Two Qubit QFT Circuit.

For the gate set, a standard Python dictionary is used to map the Qiskit quantum gate objects required to build the desired circuit to a single integer value which can be referenced in the circuit representation.

A variable length 2 dimensional Python list is used to store a quantum circuit. Here each quantum circuit is represented by a single list, where every index in this list represents a gate or wire used by the circuit. This list also implicitly stores the order of the gates in the circuit, with the gate at index one being applied before the gate at index two and so on. Additionally, each gate is further represented by a list, with the first index corresponding to an integer which, when looked up in the gate set dictionary, maps to a Qiskit quantum gate object. The second index is a further list which stores the qubit(s) the gate acts upon.

Breaking down the example in Figure 3, the circuit variable stores a list representing the two qubit QFT circuit, where the first index represents the Hadamard Gate which is applied to the first qubit, the second index represents the Controlled Phase Gate which is applied to both qubit 1 and qubit 0, the third index represents a wire which is applied to qubit 1, etc.

It is important to note that even though wires are included as part of the gate set and used in the representation, Qiskit does not explicitly include them in diagrams or store them as part of their built in representation. Therefore, they merely act as a placeholder value that allows for smaller circuits - circuits with fewer gates - to be evolved. The justification behind the use of this representation is simple, it directly mirrors how Qiskit implicitly stores QuantumCircuit objects (the structure returned by calling the .data attribute on a circuit object). Additionally, the representation drastically simplifies how the operators in the EA work, as they only need to look up and perform operations on indices in a list, and makes the process of creating a circuit object and converting circuits from one representation to another near trivial.

3.3 Quantum Algorithms

This segment of the paper sets forth the design choices for and implementation of, the quantum algorithms used by this paper. Additionally, to make comparison between the results of this paper and the benchmark results found in [10], the same circuit sizes used in that paper were used here, with two, three and four qubit variants of both quantum algorithms being implemented.

3.3.1 QFT Circuit Synthesis

To provide better encapsulation of the quantum algorithms from the main EA, the quantum circuits being used are synthesised and stored in a separate .py file, which is then imported by the main EA. Storing the quantum algorithms in this way removed the risk of a central point of failure, making the program easier to debug.

Additionally, each of the circuits evolved by the EA, the gate set (dictionary mapping the Qiskit objects to integers) and the pool of all possible gates the EA can select from (a gate set stored in the representation used by this paper as that is what each circuit is stored as) is hard-coded into this .py file. While storing the circuits this way does make the program less extensible and versatile, it saved me a significant amount of time from having to design and implement a variable length circuit creation algorithm(s). Moreover, as the circuit sizes evolved were fairly small, there was only a minute impact to the time and space complexity of these programs.

The actual implementation of the circuits was done using Qiskit's built in QFT method, which adds a QFT operator covering the corresponding number of qubits. An empty two, three and four qubit quantum circuit is initialised, the operator is applied to each circuit and then the unitary matrix representing each circuit is found and stored in an array. Note, to evolve this circuit correctly, it has to be decomposed once - which is to break up a gate or operator into its constituent gates. In this case, this involves using the Qiskit decompose() method to break up the QFT operator into the sequence of swap, Hadamard and phase gates which provide the same output.

The swaps at the end of the circuit can be ignored as the QFT appears at the end of the circuit and the re-ordering of the qubits can be done classically; however, as the QFT was the only operation included in each circuit and for sake of completeness, they are included.

3.3.2 Grover's Search Circuit Synthesis

The circuits are synthesised in the same manner as the QFT circuits, with a corresponding .py file being used to store the algorithms. Additionally, the structure and variables (gate set, set of possible gates and the circuits themselves) mirrors that of the QFT circuit file.

A prerequisite for this algorithm is the marked states in which it searches for. To satisfy this, the following marked states are stored as column vectors in a standard array variable: $|\uparrow\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ for the two qubit circuit;

$|\uparrow\rangle = \begin{bmatrix}0\\1\\1\end{bmatrix}$ and $|\uparrow\rangle = \begin{bmatrix}1\\0\\0\end{bmatrix}$ for the three qubit circuit; $|\uparrow\rangle = \begin{bmatrix}0\\0\\0\\0\end{bmatrix}$ and $|\uparrow\rangle = \begin{bmatrix}1\\0\\1\\0\end{bmatrix}$ for the four qubit circuit. These marked states are then passed into a function which creates an oracle that amplifies the provided marked state and adds it to a circuit. This oracle is then passed to Qiskit's GroverOperator() method to create a circuit with a complete diffusion operator that amplifies the specified marked state. The quantum circuits with the corresponding sizes are initialised, have a Hadamard gate appended to the start (all complete circuits for this algorithm must start with this), and then have the respective diffusion operator applied the optimal number of times (this is done via an auxiliary function). Then, akin to the QFT circuits, the unitary matrix representing the circuit is calculated and stored as an array. To get the correct gates needed to evolve the circuits, they must be decomposed (in the same manner as the QFT circuits) twice, as two types of operators have been applied, first the oracle, then the diffusion operator. Additionally, to aid with understanding and comparison, diagrams, in latex format, of each of the circuits are generated and stored in a separate directory (this is similarly done for the QFT circuits).

Furthermore, it became clear during the implementation of these set circuits that the gate sets for both algorithms (applied to the QFT circuits only to maintain consistent results) had to be limited to only the gates which were necessary to formulate the complete circuit, so any permutations of these gates were precluded from the set of possible gates. The reasons for this are investigated in the results section.

3.4 Evolutionary Algorithm

This section of the paper states and justifies the design choices and implementation of the operators, selection method and the fitness method employed by the EA.

3.4.1 Operators

Proprietary functions as opposed to built in methods were used for the operators as the corresponding Deap operators [17] did not work as intended with my representation and because it permitted me to extend the functionality of the operators. This extension increases the versatility of the program, making it easier to for anyone to extend or alter the functionality provided by these functions. Moreover, even though Deap is an open source library, defining the methods as such reduces the amount of reliance on external libraries.

The mutation operator works by finding a random integer in the range of the size of the list representing the circuit and replacing the gate currently stored at that index with a randomly chosen gate from the gate set (this could even be the same gate). Currently the pool of gates that the mutation function selects from is defined/ hard-coded in the mutation function itself, as the mutation function has to be registered with the Deap Toolbox, which does not allow for extra parameters (ones additional to those which are already defined) to be passed along to the mutation function. This has the accidental benefit of making the mutation function more modular and self-contained.

The crossover operator has been implemented in such a way that k-point crossover can be successfully executed. This operator works by selecting a specified number of indexes in the list - 2 in the case of this project - with validation to prevent the same index from being selected more than once. These indices are then sorted, with the values of all the indices between these selected indices being swapped with the values of the corresponding indices in the other circuit.

Before returning the altered circuit(s), both functions remove the fitness value associated with the individual circuit(s) they alter so that the program knows which circuits must have their fitness values recalculated.

3.4.2 Selection Method

The selection method utilised by this EA differs from the other segments of the EA, in that it is defined by a built in Deap method as opposed to a proprietary function. The standard benefits that come with using pre-built methods provide the majority of the justification for this use. For example, the module has already been tested, utilising it saves time and lines of code and it seamlessly works with other Deap modules.

The EA for this project utilises elitist tournament selection, essentially meaning that the EA selects individuals by carrying over a specified subset of the best individuals and selecting the rest through a series of tournaments, which randomly select a specified number of individuals, with the strongest individual winning and passing on to the next generation.

To this end, the tournament selection method, selTournament() is used and greatly simplifies the implementation. After registering the method with the toolbox, providing it with the population to select from, the number of individuals to select and the size of the tournaments is sufficient to use it.

3.4.3 Fitness Method

The fitness method employed by the EA works in three parts, first the individual which is passed to the function is converted into a QuantumCircuit object via an auxiliary function. The unitary matrix representing the transformation the converted circuit applies to the input qubits is found as an array of complex values. Finally, the absolute element to element difference between each element in the aforementioned array and the array representing the matrix for the goal circuit is found via a simple nested for loop. This is identical to the method employed by Williams and Gray in [18] and is represented by the following formula:

$$f(S,U) = \sum_{i=1}^{2^N} \sum_{j=1}^{2^N} |U_{ij} - S_{ij}| \tag{2}$$

where S is the matrix representation of the current circuit, U is the target matrix and $S, U \in U(2^N)$.

The EA attempts to minimise this function, reducing the difference between the target circuit and the circuits in the population. Here a fitness value of 0 represents an "optimal" circuit (one which may not be gate for gate identical yet still produces the exact same transformation on the input qubits).

In the implementation, the circuit conversion function is provided with the gate set, the number of qubits in the circuit and the circuit object itself. Using this data, the function sequentially parses through the provided circuit; for each index in the circuit's list the Qiskit quantum gate object and the qubit(s) it acts on are appended to a QuantumGate object which is initialised at the start of the function (unless the index represents a wire, as these cannot be added to or visualised by Qiskit's representation).

It should be noted that despite this function only calculating a single float value, it is required to return a tuple (holding just this single value) as Deap treats single-objective evaluation procedures as special instances of a multi-objective approach [17]. Furthermore, Deap requires a proprietary evaluation procedure to be defined (doesn't come with a pre-built evaluation function), enabling the same benefits found when defining my own functions for the other operators (increased versatility, modularity and extensibility).

3.4.4 Results Plotter Function

The Deap logbook is an object which compiles and stores the statistics which are registered with the Deap statistics module at the start of the EA. In this case, the average circuit fitness, minimum circuit fitness found and circuit size are stored in the logbook, as they provide a good overview of the general performance of the EA (show the best and average results throughout a run). This logbook provides a convenient and memory efficient method of storing and handling the data generated by the EA. To display these statistics, Matplotlib is then used as it is a library I was already comfortable with and enables the plots to be saved and altered after use. Matplotlib does this by collecting the data stored in each chapter - a labelled array of values - in the logbook and plotting it (using the .plot() and then .show() functions) across the number of generations carried out by the EA.

3.4.5 Main Function

The pseudocode for the main executable function that executes the entire EA is included below. Note to insure that the pseudocode is as general as possible, any Deap methods and libraries are precluded from the pseudocode. Preceding this function and included in the population initialisation, Deap libraries are used to register all the operators and the logbook (compiles and tracks statistics from a run of the EA) with the Deap toolbox (the brain of the module that calls and connects each of the Deap methods used by the EA). Furthermore, the Select, Fitness, Mutate and Crossover function calls represent calls by the Deap toolbox or otherwise to the function which implements the corresponding operator. The randomInt() and randomChoice() function calls equate to the Python random module's random.randint() and random.choice() functions, which create a random number and randomly select an item from a list, respectively. Finally, the elitism rate, E, is provided as an integer representing the number of circuits which are

Algorithm 1 Main Evolutionary Algorithm

```
1: procedure MAIN(G, P, M, C, E)                ▷ G is the number of generations, P is the population size
2:     Initialise Population_P                  ▷ M, C and E are the rates of mutation, crossover and elitism respectively
3:     bestSolution = ∞                         ▷ Tracks the best solution found by the EA
4:     for individual in Population do
5:         individual.fitness = Fitness(individual, S, M, Q)
6:     end for                                  ▷ S is the gate set, M is the goal matrix, Q is the number of qubits
7:
8:     for i in range 0 to Generation do
9:         offspring = Select(Population, P, T)          ▷ Selects P members from the Population
10:                                                      ▷ T is the tournament size used
11:        nextGenPopulation = offspring                 ▷ Stores a complete copy of the offspring
12:        nextGenPopulation.sort()
13:
14:        if nextGenPopulation[0] < bestSolution then
15:            bestSolution = nextGenPopulation[0]
16:        end if                                ▷ Replaces the current best solution if there is a better one
17:
18:        nextGenPopulation = nextGenPopulation[0:E:]    ▷ Passes E solutions to the next population
19:        for tuple(ind1, ind2) in offspring do         ▷ Applies the crossover operator to the population
20:            if randomInt() < C then
21:                Crossover(ind1, ind2)
22:                ind1.fitness = null
23:                ind2.fitness = null
24:            end if
25:        end for
26:        for ind in offspring do                       ▷ Applies the mutation operator the population
27:            if randomInt() < M then
28:                Mutate(ind)
29:                ind.fitness = null
30:            end if
31:        end for
32:        for ind in offspring do               ▷ Recalculates the fitness values for any altered individuals
33:            if ind.fitness is null then
34:                ind.fitness = Fitness(ind, S, M, Q)
35:            end if
36:        end for
37:                                  ▷ Fills up the rest of the next population with solutions from the offspring
38:        while nextGenPopulation.length < population.length do
39:            ind = randomChoice(offspring)
40:            nextGenPopulation.append(ind)
41:            offspring.remove(ind)
42:        end while
43:        Population = nextGenPopulation                ▷ Completely updates the population
44:    end for
45:    return bestSolution
46: end procedure
```

carried over.

3.5 Experiments Design

To mitigate the amount of external noise and attain consistent results the same base values were used for each of the experiments. The values chosen here are inline with those used in other literature in this space. The mutation rate was set to the greater value of 0.4 or the reciprocal of the size of the circuit (similar to [8]). The crossover rate was set to 0.7, which is similar to [8][19][20]. The elitism rate was set to 5% , which is similar to [10][20][21]. Both the population size and number of generations were set to 100 as it seemed like a good compromise to ensure

the experiments ran quickly but still had the time and depth to converge and explore the state space (the set of all potential circuits the EA can evolve). Finally, the tournament size was set to the lowest possible value it could take, two.

Furthermore, the 4 qubit QFT circuit was evolved in each of the experiments as it provided a suitably large state space to evaluate the impact of each parameter. Additionally, each experiment follows the same structure, tracks the same metrics and is run the same number of times.

This project executes two types of experiments, those which alter the parameters utilised by the operators, selection method and other parts of the EA, and those which implement and evaluate a multi-objective EA. The parameter experiments cover all major influences on the EA that can be altered and include the following:

- Mutation Rate: This rate can hold any value between 0 and 1 (no mutation occurs or every individual in the population is mutated), with the range of values being evaluated in increments of 0.1.

- Crossover Rate: Much alike the mutation rate, any value between 0 and 1 represents a valid rate of crossover (no crossover occurs or every individual has a subset of its genes swapped with another individual), so this range was evaluated in increments of 0.1.

- Population Size: Starting from the base value, the size is evaluated up to a population size of 700 in increments of 100. Initial executions of this test and the assumption of an exponential increase in execution time meant that population sizes in excess of 700 became sufficiently intractable to warrant execution.

- Number of Generations: Starting from the base value, the number of generations is evaluated up to a size of 700, in increments of 100. The justification behind this is identical to the aforementioned.

- Elitism Rate: This is a particularly sensitive parameter, so it was increased in sizes of 5% up to an elitism rate of 45%, providing a good range of results without reaching outlier values.

- Tournament Size: Starting from the base value, the tournament size was increased to a size of 10, in increments of 1, as this was the smallest possible increment and provided a breadth of values to examine.

It is not the intention of this project to implement nor test a multitude of multi-objective EAs, they merely provide another set of results to compare against the main EA used by this paper. To this end a single multi-objective EA which optimised circuits against both size and absolute error was implemented. The size of quantum circuits was chosen as the next objective to optimise against as shy of finding the correct circuit, it provides the most practical and applicable benefit of any objective. Moreover, for simplicity sake and to better appreciate the direct impact of both objectives, no other objectives were considered.

The structure of this multi-objective EA is fairly similar to the main EA, with the only difference being the returns of the fitness function and the selection method used (the multi-objective EA must sort and select individuals according to two equally weighted objectives as opposed to one). The modification made to the fitness function involved adding the size of the quantum circuit to the returned tuple as Deap already treats single objective fitness functions as special instances of a multi-objective fitness function.

Additionally, it became apparent that the tournament selection operator could not be altered in such a way that it would still work with the multi-objective EA and that a new method all together would have to be adopted. The fast and efficient variant of the non-dominated sorting genetic algorithm (NSGA-II) [22] was chosen as it has showed excellent results when applied to multi-objective EAs [23] and is already implemented as a pre-tested function in the Deap algorithms module [24]. None of the pre-existing structure of the EA had to be altered to implement this; the NSGA-II algorithm was registered with the toolbox, with the same parameters (naturally excluding the tournament size argument) being passed when it is called in the main function.

No additional statistics, e.g. minimum circuit size throughout the evolution, were calculated to ensure that the comparison between the two EAs remained as elementary as possible. This again was done through the Deap logbook, with the some plots and figures being displayed by the main function.

3.6 Evaluation Criteria

The complete evaluation of this project necessitates assessing the project against three criteria. Firstly unit testing is applied throughout the development process to determine whether the functional requirements of the project are being satiated. This is done by providing each function with a mock input and validating the output. Secondly, benchmark results from both [10] and [21] are used to assess the capabilities of the system. After the EA was

implemented, a comparison between our results and the results of these papers is done to determine the holistic success of the project. The amount of divergence between the results in the aforementioned papers is then used as a direct guide on the degree to which the EA and all other technical aspects of this project were successfully implemented. Finally, despite the non-functional requirements not being required, if the project is to be truly successful it must implement them to an exceptional degree. Therefore, an examination on the extent to which they are met is carried out.

3.7 Project Resources

To successfully implement the project in a way that minimises the amount of sophistication and redundant functionality, the following resources have been used:

- Jupyter Notebook: An interactive development environment [25] which has solid integration with Qiskit. Crucially, it has in-line visualisation, which enables you to graphically display quantum circuits without having to write to a file, simplifying the coding and reducing the time taken to debug or experiment with quantum circuits.

- Qiskit: An open-source software tool written by IBM for Python [26]. This is the core of the project and facilitates software that operates at the circuit level of quantum computation. Moreover, it enables quantum programs and algorithms to be both written and executed (on quantum or classical hardware).

- Deap: A third party library [27] that provides the foundation for the EA in this paper. This framework provides built-in templates for all operators, selection procedures and fitness methods in the EA. Additionally, a plethora of pre-built functionality is provided, i.e. statistics tracking [28], population initialisation, etc.

- Matplotlib: An open source Python library used for plotting data. Used within this project for graphically displaying a variety of statistics from a given run of the EA. There is great synergy with Jupyter Notebook, with the plots both able to be downloaded and altered post creation.

- Numpy, Math and Random Python Modules: Numpy is a third party library used for calculating the average, minimum and maximum fitness and size values of circuits throughout a run of the EA. Math and Random are both modules already included in the Python standard library and are used for calculations (in both the main function and the fitness function) and implementing the stochastic functionality required by the operators respectively.

4 Results

This part of the paper states and examines the results of the parameter experiments, the multi-objective EA and the single objective EA when applied to the QFT and Grover's Search.

4.1 Initial Results

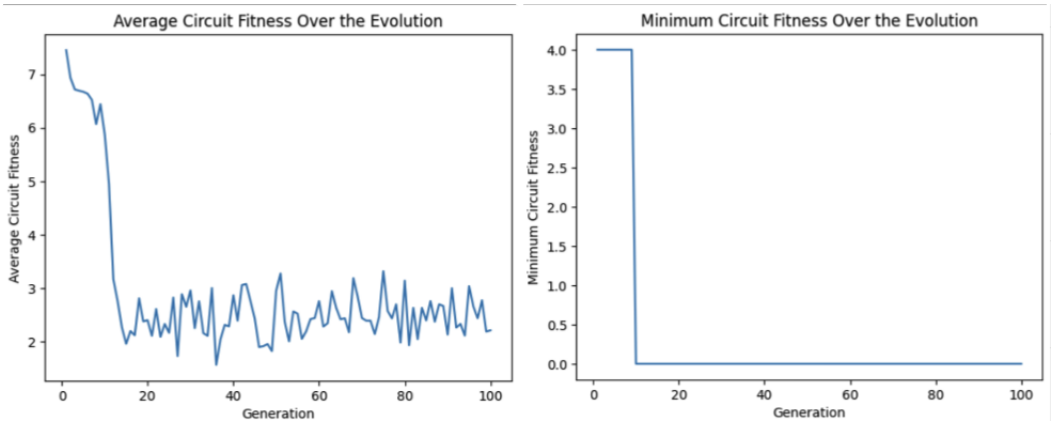


Figure 5: Initial Results For the EA When Run on a Two Qubit Mock Circuit.

The initial results of the EA when evolving mock test circuits and even the two qubit circuits for both quantum algorithms was promising, with the EA able to generate circuits that were optimal or near optimal in terms of both size and fitness. However, two things became glaringly obvious when the algorithm was first run on the larger circuits (especially the three and four qubit Grover's Search circuits), the first was that the execution time for the algorithm scaled up exponentially and quickly became unpractical; secondly, the degree to which the EA could converge on a decent size or fitness value rapidly diminished as the size of the circuit it was evolving increased.

The implementation of the solutions for these problems is outlined in the Design and Implementation segment of this paper. The inefficacy of the algorithm for larger circuits came down to its inability to correctly utilise the entirety of the gate set for a given problem. This was resulting in the EA producing circuits, like the one seen in Figure 6, that contained all of one type of gate and none of the other gates which were essential for the complete circuit. For example the circuit below should contain a Hadamard gate on each qubit, yet contains none.

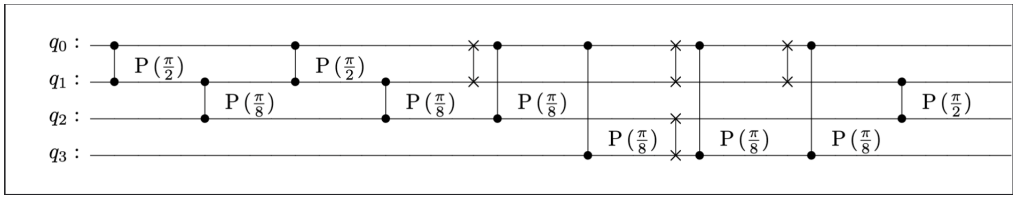


Figure 6: A Four Qubit QFT Circuit Evolved by the EA.

Additionally, this meant that the fittest circuits found by the EA (especially for larger problems) were capped at a fairly high fitness value, preventing the EA from converging or finding any remotely decent solution. This also had the side effect of promoting the proliferation of the size of a circuit and increasing the execution time. The EA would add lots of unnecessary gates in an attempt to compensate for the necessary gates not being present, which ultimately increased the execution time of the algorithm as the circuits took longer to evaluate. As can be seen in the graph below, the average fitness value and average circuit size is massively inflated and though both converge a little, they do not come anywhere close to their optimal values.

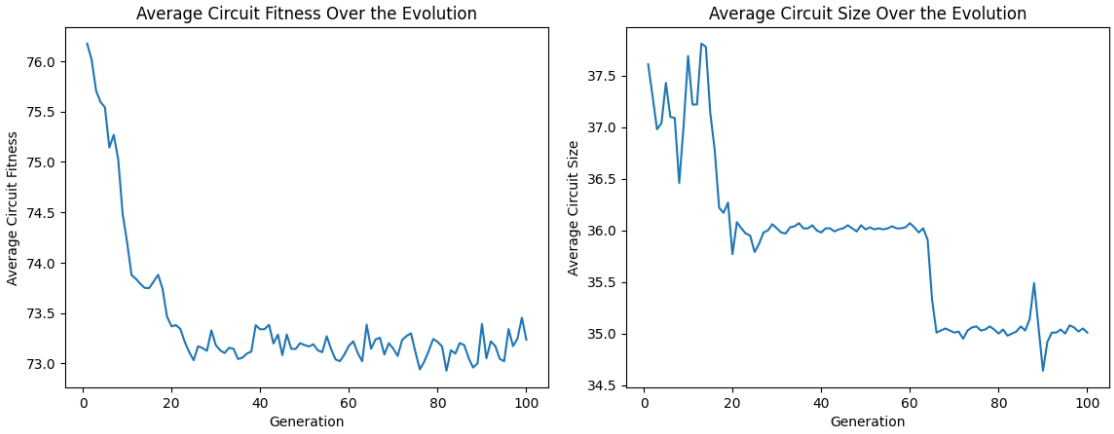


Figure 7: An Initial Fitness and Size Plot Produced by the EA for the Four Qubit QFT.

The solution to this involved reducing the size of the gate set for each algorithm such that necessary gates were no longer inhibited from appearing in solutions. Doing this greatly increased the amount of external influence that is exerted over the EA, which, even though this is in opposition with good EA practices, was essential for attaining permissible results.

4.2 Experiment Results

This portion of the paper states and evaluates the results of the parameter experiments and the multi-objective EA.

4.2.1 Parameter Experiments

Table 1. The results of each of the parameter experiments are included below, with the parameter value that lead to the lowest average fitness value being the one displayed in the table. Average Fitness, Average Size and Average Time Taken measure, over the 10 runs executed by the EA, the average best circuit fitness value found, the average best circuit size found and the average execution time respectively. For the sake of transparency, the full results of each of the parameter experiments are included as in **Appendix A**.

Experiment	Parameter Value	Average Fitness	Average Size	Average Time Taken (s)
Mutation Rate	0.8	36.76	37.8	63.51
Crossover Rate	0.6	28.62	36.9	67.27
Elitism Rate	0.45	38.14	38.2	61.28
Population Size	600	25.51	36.2	328.46
Generation Size	600	29.95	37.3	410.77
Tournament Size	4	39.17	36.8	68.08

The results from the mutation rate experiment were much higher than expected (exactly twice the base rate used in this paper and in [8]), yet not surprising in context. Such a high rate of mutation results in a faster, more dangerous exploration of the state space; however, it appears this benefited the EA and enabled it to quickly converge on a solution better than any other rate.

Furthermore, regarding the crossover rate experiment, the results were almost perfectly inline with what was anticipated (only 0.1 off of the base value used in this paper, as well the value used in [8], [19] and [20]). This enables the EA to explore, at a high rate, the solutions in the state space near (but not surrounding as this is the premise of the mutation operator) solutions which are already in the population. Additionally, this rate provided the most significant impact to the quality of circuits found by the EA.

Results for both the population and generation size show that values that were significantly larger than the baseline values were optimal for this EA. These results indicate that, if the computing power permits it, the ideal approach to quantum circuit synthesis is both broad and deep. This approach does create a pretty noticeable hike to the EA's execution time (≥ 5 times larger than every other parameter). For this project, a trade-off was made between the quality of the circuits produced and the time taken by the algorithm, as utilising both the ideal population and generation sizes lead the EA's execution to rapidly become unpractical. To this end, when collecting the results for the following sections, a generation size of 400 was used, as it provided a 33% reduction in execution time whilst also resulting in a limited impact to the overall results of the EA (as the population size was sufficiently high, the average fitness found tapered off well before 600 generations was reached).

The outcome of the tournament size experiment produced results that were inline with what was anticipated, that smaller tournament sizes, between three and five (with a size of four appearing as the optimal size), leading to the best results. In principle, smaller tournament sizes result in less selection pressure, where this is the amount of external influence exerted over the selection process, preventing too strong of a bias on fitter solutions from forming and thereby allowing the EA to adequately propagate weaker solutions (enables better exploration in favour of exploitation).

Moreover, I believe the results from the elitism rate experiment to be misleading and even though an elitism rate of 45% was found to be optimal (it was also the highest elitism rate tested), the base elitism rate of 5% was used by the EA when producing the outputs in the succeeding sections for both EAs. This is because the elitism rate is a particularly sensitive parameter and can very easily lead the EA to converge on a local optima (especially with larger values). This makes intuitive sense as carrying over near half the population without alteration into the next generation would significantly reduce the EA's exploration of the state space, whereas only carrying over a small subset maintains only a slight bias towards fitter solutions.

4.2.2 Multi-Objective EA

Table 2. To adhere to the format and benchmark results found in [10] as strictly as possible, the metrics used to assess the results of the multi-objective EA closely mirror those which they used. Here the Success Rate indicates the percentage of runs that produced a solution with a fitness value of 0. The Best Fitness metric displays the fitness value and size of the best circuit found across all the runs. Interquartile Range (IQR) expresses the distribution of the globally best fitness and size found across each of the 10 runs in the form (fitness, size). Size closeness is a novel metric I've used for this paper which measures the average margin by which the size of solutions deviates from the size of the target circuit. Each problem is represented with a three letter abbreviation, GDO for the Grover's Search and QFT naturally for the Quantum Fourier Transform, with the number to the right of the name indicating the number of qubits in the problem. The full results for the multi-objective EA are included in **Appendix B**.

Problem	Success Rate	Best Fitness	Interquartile Range	Size Closeness	Average Time Taken(s)
GDO-2	0%	(4, 3)	(0, 3)	11.5	58.56
GDO-3	0%	(17.19, 14)	(0.49, 10)	16.6	146.6
GDO-4	0%	(46.75, 108)	(10.625, 31)	33.7	2727.58
QFT-2	60%	(0, 4)	(2.83, 1)	0.3	145.70
QFT-3	0%	(4.33, 6)	(3.39, 1)	1.1	302.34
QFT-4	0%	(36.61, 9)	(8.45, 1)	2.5	516.94

Initial runs of the Grover circuits with the optimal parameters were completely intractable (particularly with the 4 qubit Grover Circuit, which took the multi-objective EA upwards of four hours to execute a single run). To resolve this and maintain consistent results the number of generations was reduced to 200 and the population size was reduced to 300 when each of the Grover circuits were evolved.

The results from the above table paint a bleak picture, with the multi-objective EA being completely unsuccessful at finding optimal circuits for all but the QFT-2 circuit (even here it only had a success rate of 60%). On the other hand, the values for this EA's IQR across all the problems suggest that the multi-objective EA was able to achieve results consistently, with there always being a reasonable deviation in both size and fitness.

Concerningly, while the size closeness for each of the QFT circuits was acceptable (single digit values), the size closeness values for each of the GDO circuits was large and out of line with what was anticipated. These results indicate that even though this EA optimises for the size of a circuit, for larger circuits, i.e. GDO-4 and GDO-3, which have a circuit size of 70 and 26 respectively, its ability to effectively do so decreases proportionally to the increase in circuit size. Furthermore, upon closer inspection of the individual solutions generated when executing each of these problems a common theme appears. For almost every problem (bar the smaller QFT-2 circuit and weirdly the larger GDO-4 circuit), this EA produced multiple solutions which had a circuit size noticeably smaller (deviating by more than two gates) than the target circuit size. Specifically, for GDO-2 a circuit with a size of one was produced and for GDO-3 on three separate runs, a circuit with a size of 6 (the optimal circuit size for this problem is 26) was evolved. Furthermore, evidence for this can be found in the table and is specifically pronounced for the GDO-3 and QFT-4 Best Fitness results, with the best GDO-3 circuit being almost half the target size. These results indicate that the multi-objective EA, instead of optimising both size and fitness, is only actually minimising the former, even at the cost of the circuit's fitness. Inspecting an individual run of the multi-objective EA on the QFT-4 problem produced the following graphs.

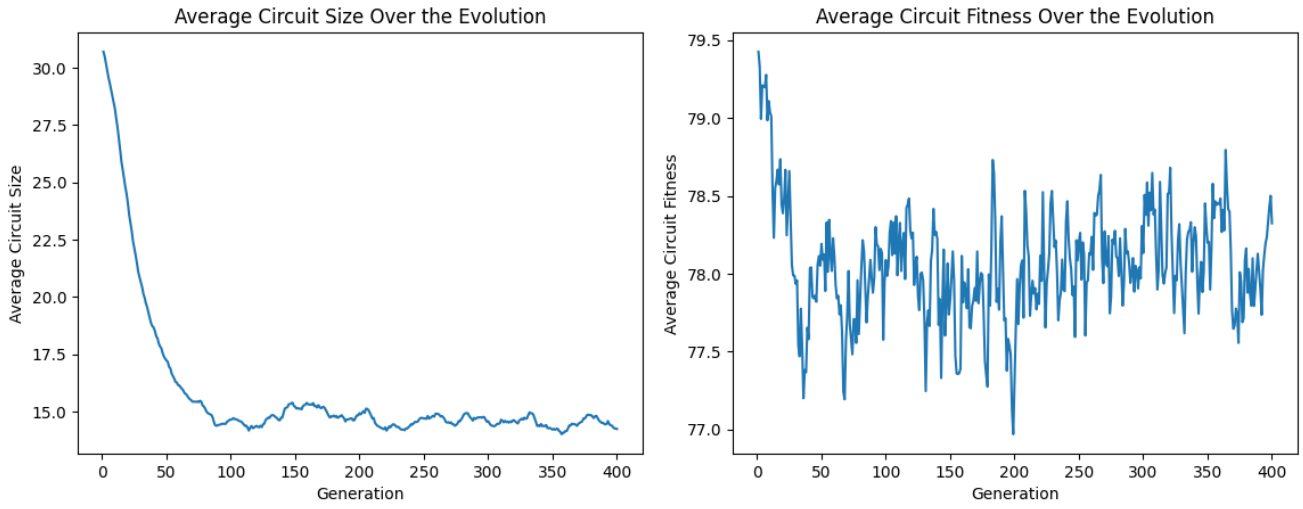


Figure 8: An Example of the Average and Best Fitness Values Found When Evolving the Four Qubit QFT.

These graphs provide further evidence for the problem found in **Table 2**. While the average circuit size is optimised as anticipated and trends downwards (though towards zero and not the optimal circuit size), the average fitness found by this EA does not converge on any value, fit or otherwise, instead oscillating quite rapidly between unfit values for the entirety of the generation. These results clearly indicate a fault within the multi-objective EA itself. The first potential cause could be with the selection method used, either the NSGA-II selection method does not work at all for this set of problems (that being quantum circuit evolution), or the objectives being optimised

are not conducive with this selection method. The second potential cause could be with the implementation of the multi-objective EA, with my assumptions about the algorithm (e.g. that it returns a sorted list of circuits) and/or my method for selecting a best circuit and carrying over elitist solutions being at fault (the NSGA-II selection method itself is pre-built and therefore, its implementation cannot be at fault).

4.3 Problem Results

Table 3. The metrics used in this table are in principle the same as those used in **Table 2.** The only difference appears in the Best Fitness and Interquartile Range criteria, which no longer show/ track the size of the best solution or the distribution of the solutions sizes respectively (this was done as this EA does not optimise circuit size). The full results for this EA are included in **Appendix C**.

Problem	Success Rate	Best Fitness	Interquartile Range	Size Closeness	Average Time Taken(s)
GDO-2	100%	0	0	0.7	130.86
GDO-3	0%	11.31	2.13	19.7	341.6
GDO-4	0%	15.75	28.69	126.6	6235.323
QFT-2	100%	0	0	2.2	270.49
QFT-3	100%	0	0	5.6	524.96
QFT-4	30%	0	22.63	16.5	1019.31

To make the results more comparable with those from **Table 2**, this EA used the same parameters for the Grover's search circuits as the multi-objective EA. This change in parameters may be partly responsible for the EA's less than stellar results for the range of GDO circuits.

The results from the above table clearly indicate the success of this EA. It is able to consistently evolve smaller circuit sizes such as the GDO-2, QFT-2 and QFT-3 circuits with a 100% success rate, showing some success on medium sizes circuits, with a 30% success rate and a larger distribution of values for the QFT-4 circuit. However, the algorithm falls short when evolving the larger GDO-3 and GDO-4 circuits and was unable to evolve a single successful circuit for either of these algorithms.

An interesting find from this table is that, on average, for the problems it successfully evolved, the main EA does not find the optimal circuit but instead finds an equivalent circuit with a larger size. Even with the smaller QFT-2 and QFT-3 circuits, which have an optimal size of just four and seven respectively, this EA produced circuits which, proportional to the target size, deviated by more than 50%. The size of the circuit does not include any wires, so these results imply that the EA struggles to exactly evolve the target circuit and instead uses multiple redundant gates to achieve the same transformation.

Furthermore, the figures for the average time taken paint the picture that the EA has a time complexity proportional to $O(2^n)$ or a time complexity that is at least near exponential. Take as example the range of average times found for the GDO circuits, with a scaling in circuit size that is not exponential, going from a size of 16 to 26 to 70 (for the GDO-2, GDO-3 and GDO-4 circuits), the average time scales well beyond what was anticipated, roughly going from an execution time of two minutes, to seven minutes to well over one and a half hours for a single execution of the EA. This clearly displays the inefficiency of the algorithm and implies a fault in the implementation or design of the EA.

Finally, the size closeness of the circuits found by the EA seem to increase proportionally to the size of the circuit being evolved. The size closeness for the smaller circuits is quite minuscule, being just 0.7 and 2.2 for the GDO-2 and QFT-2 circuits respectively; however for the larger circuits, the GDO-4 and QFT-3, the size closeness is significantly larger, being 126.6 and 16.5 respectively. The conclusion from this subset of the data is that the EA's ability to effectively evolve circuits which converge near the optimal size diminishes as the circuit size increases and that the EA is therefore not able to scale up to larger problems.

Inspecting a individual run of the main EA on the QFT-4 problem produced the graphs in Figure 9. The graphs in this figure provide visual evidence for the successes and failures of this EA, highlighting that even for intermediately sized circuits, the EA is able to converge quite well, but still not to an acceptable degree of fitness (perhaps getting stuck in a local optima, which is more likely to occur in problems with a larger state space). In spite of this, the EA is able to find an optimal circuit for medium sized quantum algorithms.

4.4 Results Analysis

A comparison of the results from both EAs used by this paper to the benchmark results produced by the swarm algorithm (Quantum Ant Programming algorithm (QAP)) used in the Quantum Program Synthesis (QPS) paper [10] leads to a few interesting results. Firstly, there are some key differences in the way results were collected and in the

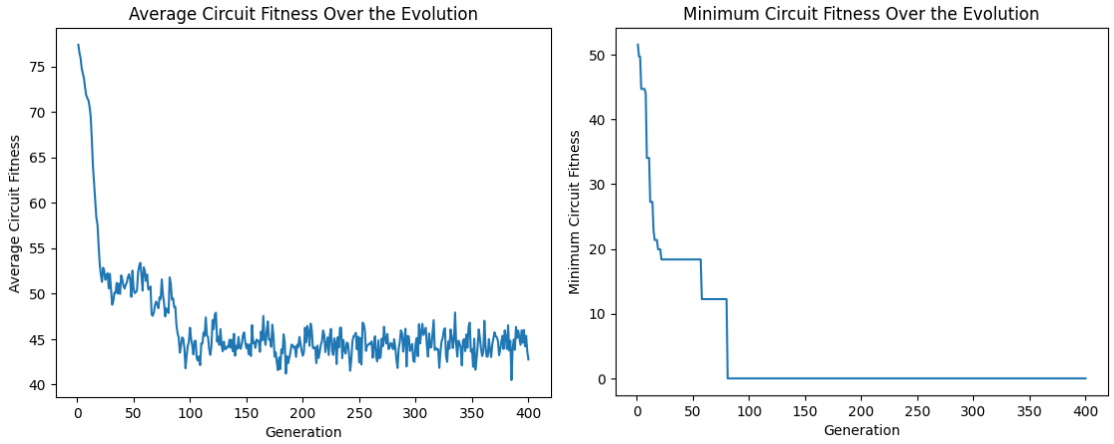


Figure 9: An Example of the Average and Best Fitness Values Found When Evolving the Four Qubit QFT.

standards of results. In the QPS paper the results presented in the tables are substantially more thorough as they ran each algorithm 100 times for each problem and let their algorithm execute up to 1,000,000 circuit evaluations (akin to the generation number used by this EA), the reason for this discrepancy comes down to the amount of the computing power available for this project being relatively low in comparison. Moreover, as their paper utilises a different fitness function, Mean Square Fidelity, it also utilises a different definition of what a "correct" solution is, accepting any solution within a 2% margin of error of a perfect fitness score as a correct solution. By contrast this paper only accepts a solution as correct if it has a perfect fitness score (0).

The biggest conclusion that can be drawn from comparing the main EA with their QAP, is that it is arguably more successful even with a differing definition of what a correct solution is. While both algorithms had a 100% success rate on QFT-2 and GDO-2 and a 0% success rate on GDO-3 and GDO-4, my EA had a 100% success rate on QFT-2 (as opposed to their 68%) and a 30% success rate on QFT-3 (as opposed to their 2%). These results create a plethora of implications but do not prove anything decisively, for example, that my EA could be generally more effective, that it could be better tuned to the problems being tackled, etc.

Their use of a different fitness method does make it harder to compare the IQR results; however accounting for this difference, their QAP can still be said to be significantly more consistent - has smaller IQR values - than my EA. Especially for the larger GDO-4 and QFT-4 circuits their QAP algorithm stays largely consistent, producing an IQR of 0 and 0.7 respectively, whereas this papers EA becomes massively inconsistent, producing an IQR of 28.69 and 22.63 respectively. For the other circuits evaluated both algorithms have an analogous IQR value.

As the results from [10] do not track size closeness or the average time taken by the algorithm, not much can be done to compare the efficiency or the deeper effectiveness of both algorithms.

A comparison of the multi-objective EA with the QAP algorithm is not even close, with the QAP algorithm being massively more successful on the whole as the only circuit the multi-objective EA had any success on was QFT-2. Additionally, even comparison with their Random Quantum Ant Programming algorithm (R-QAP), which is a random search equivalent of QAP, shows that the multi-objective EA massively under performs both of their benchmark standards for these problems.

As neither of their algorithms optimise for circuit size, comparison between the consistency of QAP and the multi-objective EA is only done on the basis of the IQR of the fitness values found. Doing this, it can be said that both algorithms have a roughly similar distribution of results, both for the smaller circuits and when scaled up to the larger circuits. The multi-objective EA, for QFT-2 and QFT-3, has an IQR of 2.83 and 3.39 respectively, whereas QAP has an IQR of 0 for both; however, for the rest of the circuits there is very little discrepancy between the IQR values found.

Carrying out an inter-comparison between the results of the multi-objective EA and the results of the main EA used in this paper results in the following conclusions. The first is that the main EA is unequivocally more successful than its multi-objective counterpart, the reasons behind this are explored in section **4.2.2**. Secondly, the main EA is more consistent on the problems that it was successful on; however, the multi-objective EA becomes more

consistent when scaled up to the larger GDO-3, GDO-4 and QFT-4 circuits, in terms of both circuit size and fitness. Thirdly, the multi-objective EA, for all circuits bar GDO-2 and GDO-3, produced circuits which had a size closer to the target size (lower size closeness value), this is almost certainly due to the multi-objective EA over optimising against circuit size. Finally examining the average time taken criteria for both algorithms, it can be said that the multi-objective EA is roughly twice as fast as the main EA, but still scales up with a similar time complexity (that being exponential or near exponential). As the selection method has the largest influence over the time complexity and execution time of the algorithm, this and the multi-objective algorithm's superior consistency on the larger circuits can be attributed to it using a different selection method (NSGA-II as opposed to tournament selection).

5 Testing

This part of the paper details the design and implementation of a range of tests used to ensure the project meets its functional requirements.

Black box testing which evaluates the functionality of each of the proprietary functions defined within the EA is used to ensure that the program functions as intended and meets its functional requirements. Moreover, as the EA only contains a single unit tested main executable (it does not have multiple software components interacting with each other), no integration or interface testing was carried out.

To this end the Python unittest module was used as it provides a stable testing framework and the necessary functionality required to create both the test cases and the test suite with ease. The individual unit tests were designed to fit the standard structure required by the unittest module [29] and to test the modules through the full range of potential inputs (boundary, erroneous and valid inputs). For each test function, the same base conditions and environment was used to maintain consistency across each of the test functions and to minimise the amount of external variables that could potentially hinder results. Additionally, a test suite was created to simplify the unit testing process and provide a single point of execution for each of the unit tests.

The following unit tests were carried out:

- Circuit Fitness: A range of valid, invalid, and very large/small circuits are provided to this function, along with the correct fitness output to affirm correct functionality. Ensures that requirement **2.2.7** is satisfied.
- Circuit Size: Akin to the above, a range of circuits are applied, along with their correct size to confirm the function operates correctly. Used as part of the main function, so partly validates requirement **2.2.1**.
- Circuit Conversion: This test function is provided with a series of invalid, valid and very large/small circuits to ensure they are correctly converted into Qiskit QuantumCircuit objects. Validates requirement **2.2.6**.
- Crossover: This function is provided with two mock parent solutions and a range of valid, invalid and boundary indices which represent the subset of genes to swap between parents, establishing that this function works as intended. Validates requirement **2.2.5**.
- Mutation: Akin to the above unit test, a mock circuit and randomly chosen index is provided, validating that the operator alters genes as anticipated. Validates requirement **2.2.4**.
- Random Choice: A standard Deap module is used to initialise the population, so the function for randomly choosing a gate from the gate set is the only portion of the initialisation process that must be tested.

Furthermore, there are a number of modules or operators that are provided by either the Deap (the initialisation procedure, selection procedure and data logging procedure) or the Matplotlib (the data plotting procedure) libraries. These modules are already validated and tested, so no unit testing was carried out on them. These modules account for requirements **2.2.2**, **2.2.3** and **2.2.9**.

Due to the stochastic nature of the EA and all of its constituent parts it is quite impossible to test and validate the outputs of the main executable function; therefore, the success of all of the aforementioned unit tests indicates the success of the EA - ensuring requirement **2.2.1** is met.

6 Description of the Final Product

All in all, this project has designed and implemented a simple EA for automatically designing quantum circuits. Implementing a basic mutation and crossover operator for altering the population and a basic tournament selection

method to select new populations, this EA uses a series of pre-built circuits (which are synthesised in an external file) for the Grover's search and QFT algorithms to guide the search and design the circuits. A visualisation library is then used to graphically display the results of each run.

To optimise this EA, a series of experiments were carried out, evaluating the impact of each of the parameters which are used by the EA, and their results were stored. A multi-objective EA was then implemented which sorted against both the size and fitness of a circuit. Finally, a series of unit tests are used to ensure that the product met the functional requirements defined in section **2.3**.

7 Evaluation of the Final Product

Overall, the product is successful in its main mission, that being implementing an EA to automatically design quantum circuits, as it is able to effectively evolve, with a high rate of success, smaller quantum circuits such as the two, three and four qubit QFT circuits and the two qubit Grover's Search circuit. Furthermore, the EA presented in this paper is able to outperform, on a range of quantum circuits, the benchmark results in the QPS paper, which provided the standards for this paper. This provides the bulk of the technical advantages of this solution, as the EA provides an incremental improvement over the QAP algorithm used in the comparison paper and has the added benefit of being better studied and accepted in surrounding literature when compared to the QAP swarm algorithm.

In relation to the functional and non-functional requirements for the product, the final product successfully implemented each of the functional requirements designed and outlined at the start of the paper, completely implementing each aspect of the EA (**2.2.1-2.2.7**), completely synthesising each of the quantum algorithms with the sizes inline with the those used in the QPS paper (**2.2.8**) and constructing a way to graphically display the results of the EA with success (**2.2.9**). Moreover, the final product only falls short of a single non-functional requirement, **2.3.3**, as while it is able to execute completely, both the main EA and the multi-objective EA cannot reasonably be said to be efficient, as they both scale up near exponentially and would be in practical for use on problems any larger than the ones examined in this paper (even being ineffective for some of the larger problems investigated by this paper).

The final product utilised and executed a range of experiments which were partially successful. The parameter experiments were successful and provided an optimal range of parameter values to guide the main EA. The multi-objective EA experiment was largely unsuccessful as a poor implementation prevented it from functioning as intended and from providing a suitable point of comparison with the main EA.

One of the more serious limitations of the product is that the results collected for both the EA and multi-objective EA were not as thorough as they ought to have been (especially when comparing the number of runs and executions to those collected in [10]). Collating the results to the same extent as [10] was computationally intractable in a reasonable time. Furthermore, the product is still limited to hard-coded circuits for specific algorithms with a specific size, providing little to no generalised behaviour which would be required to make the project truly usable in wider applications.

An unintended result of this approach is that the main EA regularly produces alternative circuits that provide the same transformation. These circuits are larger in size but prove that the EA has the utility of investigating ulterior circuits which may not have been considered before and which could potentially be more practical.

8 Holistic Critical Assessment of the Project

This project has shown that a simple EA, with the help of a few external libraries, is able to effectively and efficiently evolve the circuits for two different quantum algorithms whose size varies from two to four qubits. This is not a novel accomplishment, with many papers, [19], [20], [21] and many more, some as old as 25 years old [18], accomplishing this same feat. Where this project differs is in the methods it employs within the EA, the use of tournament selection (I did not find this used in any surrounding literature) and the combination of parameter values used for each operator/ variable used in the EA. Additionally, the representation used to store the quantum circuits is usually the standout feature of papers on this topic, with my representation mirroring the representation found in the Qiskit library, also acting as a way to distinguish this project. Finally, this project has been one of the first to be applied to and collect results for, the benchmark problems outlined in [10].

The source code for this project is accessible and hosted on a public repository to minimise the effort required to extend or reproduce the project (any and all project files are hosted on the following GitHub repository: `https://github.com/BenjaminTheron/ECM3401_Dissertation`), which is not something that is found with many other papers in this field.

This paper has investigated the impact of each parameter used by EA in order to isolate and quantify their roles in relation to the performance of the EA, which is not something that has been explored or done much in this field. Additionally, when investigating the results of both the multi-objective EA and the main EA, novel criteria, which had not been included in the benchmark results, were added so that a more exhaustive comparison of results could be carried out between the EAs used in this paper, and any future algorithms which implement these same benchmark problems. Furthermore, while the results from this paper do not supersede those found in [10], they certainly provide a stronger baseline for future papers covering these problems to aim towards, especially if any of these papers are implementing EAs similar in principle to the one(s) used in this paper. The exact incremental improvements provided by this EA over the QAP algorithm can't be examined too deeply as this paper evaluates the algorithm against additional criteria and the algorithms in both papers are themselves fundamentally different. However, the overall efficacy with which the QFT and Grover's search problems can have circuits automatically designed for them can and is analysed in this project.

On the other hand, this project was particularly rigid with its scope and neglected to implement any quantum algorithms or evaluate any circuit sizes outside of those defined in the QPS benchmark. Extending the scope of the project and covering a broader range of problems and sizes may have warranted this project more research significance.

As stated at times prior, the results and experiments carried out as part of this project fall short of the level of robustness found if most other modern literature in this area, hindering the validity of this paper as the results, conclusions and trends drawn from this paper are not as solid as those found in other papers, even those with weaker results.

Finally, this project does not make substantial strides in answering or working towards the big questions in this area, i.e. can a generalised approach be made such that given any problem or circuit an algorithm is able to correctly evolve an optimal circuit. These questions and the steps which must be taken to answer these questions stayed out of the scope of this project (i.e. a generalised gate set was not used due to computational limitations) but would have supplied a higher level of research utility to this project.

8.1 Further Work & Research

After a basic implementation of the EA and quantum circuit synthesis programs I had the opportunity to present my progress to both my dissertation supervisor and Fujitsu Research. The feedback from both parties provided not only necessary constructive criticism but highlighted three key areas in which the project could be extended in order to achieve more practical and significant results. These were:

- The current fitness function utilised in this project is not sophisticated enough. A more thorough fitness function, such as the Mean Square Fidelity function used in [10], which tracks circuit phase and has a more practical formula as opposed to just finding the absolute difference of the circuits should be used to extend this project.

- The introduction of external noise (random signal fluctuations in a circuit) into the circuit could provide a way to reduce the number redundant gates in a circuit and make the solutions it produces more practical.

- Utilising a more sophisticated multi-objective EA, which optimises not only against one other parameter, but a range of different objectives simultaneously. For example, optimising against circuit fitness, size and depth at the same time to produce the most practically applicable circuits possible.

There are also a few areas of future research outlined in the QPS paper, [10], which were not covered in this paper, yet still remain applicable to this project, these are:

- To investigate the use of sub-systems to generate, as linear combinations of the gate set, useful single qubit gates to facilitate the approximation of circuits for bigger problems.

- To extend the EA in this paper (or any algorithm for automatically designing quantum circuits) to n-qubit problems. This would enable the EA to learn quantum algorithms outside of hard-coded circuits and become generally practical.

Further work on the products defined in this paper, both the multi-objective EA and the main EA, will involve fixing the implementation of the multi-objective EA such that it is able to correctly optimise against both parameters. This work will also involve investigating the reasons behind and remedying (if possible) the exponential

time complexity of both algorithms to enable them to be practically applied to larger circuits (or to come to the conclusion that a different approach is needed for larger circuits). In addition, an in-depth analysis of the EAs evolution of the three and four qubit Grover's search circuits will be carried out to determine where the EA is going wrong in the evolution and then if possible constructing a solution to resolve the issue.

Quantum computers are typically used for computationally heavy tasks such as particle simulations and advanced calculations, so most research done on this area tests these systems on larger scale high performance machines. Additionally, with the findings and resources from this project being made available to Fujitsu Research, who have access to and utilise large scale supercomputers (they currently have the fourth strongest supercomputer in the world [30][31]), it would make sense to use high performance systems of a semi-similar magnitude to test the limits of the EA and its parameters, thereby ensuring the EA is as applicable as possible for their use.

Finally, a range of experiments can be executed to further validate the EA, these include:

- Executing the parameter experiments in more depth to further isolate their optimal values.
- Experimenting with the operators used by the EA, for example, by using different selection methods, multi-point mutation, K-point crossover, etc.

9 Conclusion

Bringing this paper to a terminus, an evolutionary algorithm with the ability to automatically design quantum circuits with a range of qubit inputs for the quantum Fourier transform and Grover's algorithm has been successfully designed and implemented.

Starting by providing the motivation and prerequisite background knowledge required to understand the task at hand, this paper then moves on to outline the functional and non-functional requirements of the project, additionally providing an explanation of the background and requirements for both of the quantum algorithms tackled by this paper. Furthermore, the design and implementation of the EA, the quantum algorithms and both the parameter experiments and the multi-objective evolutionary algorithm used in this paper are outlined. To aid with the understanding of this area, a UML function diagram of the technical architecture is provided, along with pseudocode for the main function used by the EA. Before moving on to the results collected for the EA, parameter experiments and the multi-objective EA, a description of the project resources and the evaluation criteria required by the project is provided. A complete analysis and comparison of these results to the benchmark results is then supplied. The unit tests used to guarantee the functional requirements, a description and an evaluation of the final product are included, with a critical assessment of the entire project, which included the future work to be done to extend the project, being provided.

This project has provided a catalogue of results for a range of benchmark problems provided in [10], even surpassing these benchmark results and the capabilities of their QAP algorithm. Additionally, these results were assembled according to a more exhaustive set of criteria to enable more thorough comparison and a stronger baseline for further work. In that regard, future work in this area will be focused on expanding the capabilities of the EA through various methods and experimenting with the methods and parameters employed by the EA.

This paper utilises and develops upon the results from prevalent literature in the area and with there remaining much more work required to progress not only the field of automated quantum circuit design but also the wider domain of quantum computation. In spite of these fields still being in a state of infancy, they retain the promise to profoundly reshape our contemporary world, revolutionising modern computation and propelling humanity onward.

References

[1] A. Shilov, "Tsmc roadmap update: N3e in 2024, n2 in 2026, major changes incoming," [Online]. Available: https://www.anandtech.com/print/17356/tsmc-roadmap-update-n3e-in-2024-n2-in-2026-major-changes-incoming (visited on 11/07/2023).

[2] F. Zhu, P. Xu, and J. Zong, "Moore's law: The potential, limits, and breakthroughs," *Applied and Computational Engineering*, vol. 10, pp. 307–315, Sep. 2023. DOI: 10.54254/2755-2721/10/20230038.

[3] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM review*, vol. 41, no. 2, pp. 303–332, 1999.

[4] I. Kassal, J. D. Whitfield, A. Perdomo-Ortiz, M.-H. Yung, and A. Aspuru-Guzik, "Simulating chemistry using quantum computers," *Annual review of physical chemistry*, vol. 62, pp. 185–207, 2011.

[5] J. Wall, "Ice ice baby — why quantum computers have to be cold," 2023. [Online]. Available: https://medium.com/the-quantum-authority/ice-ice-baby-why-quantum-computers-have-to-be-cold-3a7f777d9728 (visited on 11/07/2023).

[6] E. Schrödinger, "Die gegenwärtige situation in der quantenmechanik," *Naturwissenschaften*, vol. 23, no. 48, pp. 807–812, 1935. DOI: 10.1007/BF01491891.

[7] M. Mitchell, *An Introduction to Genetic Algorithms*. The MIT Press, 1998, ISBN: 9780262280013. DOI: https://doi.org/10.7551/mitpress/3927.001.0001.

[8] T. Yabuki and H. Iba, "Genetic algorithms for quantum circuit design –evolving a simpler teleportation circuit–," 2000. [Online]. Available: https://api.semanticscholar.org/CorpusID:16687378.

[9] J. F. Miller, P. Thomson, T. Fogarty, *et al.*, "Designing electronic circuits using evolutionary algorithms. arithmetic circuits: A case study," *Genetic algorithms and evolution strategies in engineering and computer science*, pp. 105–131, 1997.

[10] T. Atkinson, A. Karsa, J. Drake, and J. Swan, "Quantum program synthesis: Swarm algorithms and benchmarks.," in *Genetic Programming*, S. L., H. T., L. N., R. H., and G.-S. P., Eds., vol. 11451, 2019, pp. 19–34, ISBN: 978-3-030-16670-0. DOI: 10.1007/978-3-030-16670-0_2.

[11] C. H. Bennett, E. Bernstein, G. Brassard, and U. Vazirani, "Strengths and weaknesses of quantum computing," *SIAM journal on Computing*, vol. 26, no. 5, pp. 1510–1523, 1997.

[12] IBM, "Grover's algorithm," 2024. [Online]. Available: https://learning.quantum.ibm.com/tutorial/grovers-algorithm#full-grover-circuit.

[13] X. Dong, B. Dong, and X. Wang, "Quantum attacks on some feistel block ciphers," *Designs, Codes and Cryptography*, vol. 88, Jun. 2020. DOI: 10.1007/s10623-020-00741-y.

[14] M. A. Nielsen and I. L. Chuang, *Quantum computation and quantum information*. Cambridge university press, 2010.

[15] IBM, "Qft," [Online]. Available: https://docs.quantum.ibm.com/api/qiskit/qiskit.circuit.library.QFT#num_qubits (visited on 03/01/2024).

[16] J. Van Gael, "The role of interference and entanglement in quantum computing.," *Semantic Scholar*, 2005.

[17] DEAP, "Operators and algorithms," 2024. [Online]. Available: https://deap.readthedocs.io/en/master/tutorials/basic/part2.html.

[18] C. P. Williams and A. G. Gray, "Automated design of quantum circuits," in *Quantum Computing and Quantum Communications*, C. P. Williams, Ed., 1999, pp. 113–125, ISBN: 978-3-540-49208-5. DOI: 10.1007/3-540-49208-9_8.

[19] P. Marek, L. Martin, K. Pawel, *et al.*, "A hierarchical approach to computer-aided design of quantum circuits," in *Electrical and Computer Engineering Faculty Publications and Presentations. 228.*, 2003, pp.201–209.

[20] S. Satsangi and C. Patvardhan, "Design of reversible quantum equivalents of classical circuits using hybrid quantum inspired evolutionary algorithm," in *2015 IEEE International Advance Computing Conference (IACC)*, 2015, pp. 258–262. DOI: 10.1109/IADCC.2015.7154709.

[21] V. Potoček, A. P. Reynolds, A. Fedrizzi, and D. W. Corne, "Multi-objective evolutionary algorithms for quantum circuit discovery," *arXiv preprint arXiv:1812.04458*, 2018.

[22] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan, "A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: Nsga-ii," in *Parallel Problem Solving from Nature PPSN VI*, M. Schoenauer, K. Deb, G. Rudolph, *et al.*, Eds., 2000, pp. 849–858, ISBN: 978-3-540-45356-7. DOI: 10.1007/3-540-45356-3_83.

[23] H. Ma, Y. Zhang, S. Sun, T. Liu, and Y. Shan, "A comprehensive survey on nsga-ii for multi-objective optimization and applications," in *Artificial Intelligence Review*, vol. 56, 2023, pp. 15 217–15 270, ISBN: 1573-7462. DOI: 10.1007/s10462-023-10526-z.

[24] Deap, "Algorithms," 2024. [Online]. Available: https://deap.readthedocs.io/en/master/api/algo.html.

[25] Jupyter, "Jupyter," 2024. [Online]. Available: https://jupyter.org.

[26] IBM, "Qiskit," 2024. [Online]. Available: https://www.ibm.com/quantum/qiskit.

[27] DEAP, "Distributed evolutionary algorithms in python," 2024. [Online]. Available: https://github.com/deap/deap.

[28] DEAP, "Computing statistics," 2024. [Online]. Available: https://deap.readthedocs.io/en/master/tutorials/basic/part3.html.

[29] Python, "Unittest - unit testing framework," 2024. [Online]. Available: https://docs.python.org/3/library/unittest.html.

[30] Fujitsu, "Fugaku supercomputer," 2024. [Online]. Available: https://www.r-ccs.riken.jp/en/fugaku/about/.

[31] Top500, "Top500 list - november 2023," 2023. [Online]. Available: https://www.top500.org/lists/top500/list/2023/11/.

A Experiment Results

This appendix stores the full results gathered from each of the parameter experiments.

A.1 Mutation Rate Results

Mutation Rate	Average Fitness	Average Size	Time Taken
0	68.38732	37.6	65.84928
0.1	47.74707	37.5	67.26709
0.2	51.51779	37.7	68.83766
0.3	47.96385	37.7	69.74807
0.4	40.57431	37.1	69.17736
0.5	48.96277	38.0	72.76347
0.6	39.17851	37.9	71.89232
0.7	39.79024	36.1	67.79921
0.8	36.76097	37.8	63.50608
0.9	45.86529	36.9	63.73299
1	43.05381	37.2	70.42087

A.2 Crossover Rate Results

Crossover Rate	Average Fitness	Average Size	Time Taken
0	50.8287	37.4	52.04183
0.1	44.38561	38.2	59.47307
0.2	41.9043	36.3	61.7974
0.3	43.38597	37.2	61.24926
0.4	45.90651	37.6	64.39681
0.5	58.397	38.5	67.24106
0.6	28.62146	36.9	67.2694
0.7	41.74044	37.0	68.82408
0.8	36.24031	37.2	72.83487
0.9	48.20729	37.3	73.71881
1	53.82839	37.4	74.15954

A.3 Elitism Rate Results

Elitism Rate	Average Fitness	Average Size	Time Taken
0.0	58.36186	37.3	78.66427
0.05	47.82127	38.6	71.29744
0.1	48.431	38.9	70.1448
0.15	46.72739	37.7	69.52391
0.2	43.56609	38.6	70.96409
0.25	54.28294	36.9	69.21527
0.3	47.64247	37.8	70.5539
0.35	45.82209	37.7	69.58524
0.4	42.20378	37.8	68.47481
0.45	38.14088	38.2	61.27506

A.4 Population Size Results

Population Size	Average Fitness	Average Size	Time Taken
100	41.894	37.6	56.48586
200	33.73278	37.6	112.64626
300	33.15193	37.1	167.08445
400	31.4968	37.5	222.82866
500	29.14217	37.1	277.95557
600	25.50674	36.2	328.46339
700	27.82285	36.9	393.99013

A.5 Generation Size Results

Generation Size	Average Fitness	Average Size	Time Taken
100	37.39029	36.6	72.98781
200	37.01585	37.3	143.86876
300	44.84971	38.0	214.39054
400	35.84072	37.4	272.52634
500	38.80207	37.4	327.83286
600	29.95289	37.3	410.76566
700	43.35695	37.3	413.22458

A.6 Tournament Size Results

Tournament Size	Average Fitness	Average Size	Time Taken
2	42.50783	37.9	80.37647
3	47.09225	37.5	69.95777
4	39.17339	36.8	68.07717
5	51.6921	38.1	69.78818
6	45.86033	37.6	70.14978
7	44.74127	37.2	69.26872
8	38.86895	38.4	70.61592
9	44.43511	38.5	71.31498
10	50.49051	36.6	68.25154

B Multi-Objective Evolutionary Algorithm Results

This appendix stores the complete and sorted results of the multi-objective evolutionary algorithm.

QFT-4		
Best Fitness	Best Size	Time Taken
(36.61268155043542, 9)	9	505.90174
(39.920921333248266, 12)	13	520.36602
(40.920122083051226, 10)	10	527.75976
(41.83372070375584, 10)	11	520.24143
(43.212103874167596, 9)	9	523.10392
(44.85799023518293, 8)	8	497.24483
(47.605639318267755, 10)	11	518.60046
(49.35167424272156, 9)	9	521.8035
(49.98919109150435, 9)	10	511.39365
(51.70244836073057, 9)	9	523.04316

QFT-3		
Best Fitness	Best Size	Time Taken
(4.329568801169573, 6)	6	308.10089
(7.999999999999999, 6)	7	320.54368
(7.999999999999999, 6)	6	286.95764
(7.999999999999999, 6)	7	284.87465
(7.999999999999999, 7)	7	293.29894
(7.999999999999999, 7)	7	299.16751
(11.391036260090292, 5)	5	318.45516
(11.391036260090292, 5)	5	314.37991
(11.391036260090292, 5)	6	298.62692
(12.329568801169577, 6)	7	299.01778

QFT-2		
Best Fitness	Best Size	Time Taken
(0.0, 4)	4	144.28184
(0.0, 4)	4	148.40474
(0.0, 4)	4	145.25223
(0.0, 6)	6	145.86468
(0.0, 4)	5	143.88727
(4.898587196589412e-16, 8)	7	150.23733
(2.8284271247461894, 4)	2	143.7797
(2.8284271247461894, 3)	3	143.30835
(2.8284271247461894, 3)	5	142.37549
(2.8284271247461894, 3)	3	149.65469

GDO-4		
Best Fitness	Best Size	Time Taken
(46.749999999999964, 108)	109	2196.60239
(47.25000000000003, 102)	103	2485.16615
(49.875000000000014, 124)	124	2277.57841
(57.25, 101)	102	3169.7135
(59.24999999999998, 92)	91	3015.15921
(60.25000000000001, 85)	85	3167.57255
(60.26587296526008, 120)	120	3143.77589
(60.499999999999986, 93)	93	2580.64216
(61.70521279348033, 90)	90	2917.27866
(62.62499999999998, 122)	122	2295.30371

GDO-3		
Best Fitness	Best Size	Time Taken
(17.1923881554251, 14)	14	130.58934
(17.1923881554251, 10)	11	137.80912
(17.1923881554251, 6)	7	134.47655
(17.192388155425103, 4)	5	135.9211
(17.192388155425103, 6)	7	131.71052
(17.192388155425103, 12)	12	134.49405
(17.192388155425103, 6)	7	135.03567
(17.677669529663667, 20)	21	135.03605
(17.677669529663675, 8)	9	127.27547
(17.677669529663678, 8)	9	131.73777

GDO-2		
Best Fitness	Best Size	Time Taken
(3.999999999999997, 4)	5	57.47773
(3.999999999999997, 4)	5	58.2564
(3.9999999999999973, 8)	8	58.81084
(3.9999999999999973, 4)	4	63.64284
(3.9999999999999982, 6)	6	58.23003
(3.9999999999999982, 9)	10	58.93508
(3.999999999999999, 3)	3	57.81511
(3.999999999999999, 3)	4	58.25781
(3.999999999999999, 3)	3	56.04524
(4.0, 1)	2	58.10082

C Evolutionary Algorithm Results

This appendix stores the complete and sorted results of the main evolutionary algorithm.

QFT-4		
Best Fitness	Best Size	Time Taken
1.098600586264493e-14	24	987.90323
2.1294267723474965e-14	23	1024.80635
3.88484893834573e-14	29	1096.08346
21.98178917389329	24	943.37712
22.627416997969487	24	972.57139
22.627416997969487	30	1117.64071
22.62741699796949	22	852.19143
22.62741699796949	23	963.93056
25.7243627352524	28	1150.62687
25.72436273525241	28	1083.93017

QFT-3		
Best Fitness	Best Size	Time Taken
0.0	9	505.16232
0.0	9	445.65748
0.0	9	458.12122
0.0	13	549.25624
0.0	15	504.9037
0.0	15	537.85856
6.927648449883945e-16	13	564.48148
3.2005983557621614e-15	13	548.08442
3.2005983557621614e-15	15	593.98328
3.841899472099148e-15	15	542.05833

QFT-2		
Best Fitness	Best Size	Time Taken
0.0	4	223.33623
0.0	4	297.11548
0.0	4	221.79607
0.0	6	292.50112
0.0	6	289.9129
0.0	6	274.99546
0.0	8	272.22873
0.0	8	285.44186
0.0	8	294.85343
0.0	8	252.71179

GDO-4		
Best Fitness	Best Size	Time Taken
15.750000000000105	198	6278.42812
15.750000000000124	182	3663.78685
22.250000000000085	184	4909.8102
32.750000000000036	199	6457.99128
33.50000000000001	190	4732.77741
40.87499999999997	202	6331.48182
41.62500000000006	210	7246.30632
50.93750000000001	201	8791.36086
52.499999999999986	194	6749.74051
56.999999999999915	206	7191.53855

GDO-3		
Best Fitness	Best Size	Time Taken
11.31370849898476	46	344.54168
11.313708498984763	44	327.48938
11.313708498984766	44	330.0576
11.313708498984768	45	336.12663
12.727922061357862	50	367.72106
12.727922061357862	53	382.00517
13.4350288425444	45	333.8748
13.435028842544408	45	350.79852
14.84924240491748	38	294.7907
14.849242404917481	47	348.61523

GDO-2		
Best Fitness	Best Size	Time Taken
2.4492935982947044e-16	17	129.38154
2.4492935982947044e-16	18	131.79939
2.449293598294705e-16	15	132.81266
2.449293598294705e-16	15	129.44024
2.449293598294705e-16	16	126.1415
2.449293598294705e-16	16	131.81986
2.449293598294705e-16	17	133.59801
2.4492935982947054e-16	17	127.61712
4.898587196589409e-16	16	132.17121
4.898587196589409e-16	16	133.86058