# Self-Hosted Quantum Program Optimization

Liam Heeger (lheeger@u.rochester.edu)

Department of Computer Science
University of Rochester

Supervised by:
Sreepathi Pai
Ted Pawlicki

May 1, 2020

# Chapter 1

# Abstract

We propose a system to optimize quantum programs using quantum computing hardware to augment the optimization process. Current limitations of quantum computing hardware constrain the maximum program length and memory resources available to programs. These limitations can be overcome in many cases when optimization of the quantum program is performed. Such a process can increase the usability of the given hardware by decreasing the program's resource usage. This optimization procedure, given a quantum program and target quantum hardware description, formulates a constraint satisfaction problem. We solve the constraint problem given using quantum algorithms. The result is a resource optimized quantum program for the desired hardware. Using quantum algorithms we can realize speedups in compilation runtime over similar classical optimization approaches. Through experimentation, our implementation shows that runtime speedup is possible and that our constraint model outperforms other common approaches to quantum program optimization.

# Contents

# Chapter 2

# Introduction

As quantum computing hardware advances and improves each year, the need to accommodate larger problems at low cost will become apparent. In order to reduce resource usage by quantum programs on quantum computers, we need to optimize these programs to consume fewer resources. The resource we want to minimize usage of are similar to those we wish to reconcile in classical computing program optimization: time and space.

We begin by prefacing the motivations for this work (section 2.1), with a brief discussion of prior schemes for optimizing quantum programs. We then introduce several important concepts that are common to quantum computing and notation relevant to this report (section 2.2). This introduction ends with a discussion of background and prior work in this area (chapter 3). The following chapter, chapter 4, details the architecture of the optimization process that has been devised. In chapter 5, the quantum program optimization solver is discussed in detail. This is succeeded in chapter 6 by a formalization of the constraint model used in the program optimization procedure. Future work to be pursued follows in chapter 8 with a conclusion of the report in chapter 9. Lastly, an appendix displaying a flow-chart diagram of the system architecture can be found in chapter 10.

## 2.1 Motivation

Quantum computers are known to be able to solve some problems quicker than classical computers. An example of such a problem may be unordered database search. A classical computer can solved by a linear search in runtime $\mathcal{O}(n)$. This problem can be solved a quantum algorithm called Grover's algorithm [1]. Grover's algorithm for database search can perform unordered lookup in quadratically less time than a classical algorithm. However, like classical computers, quantum machines are bound by the resources they have available to them. At the time of writing this is especially concerning as state-of-the-art quantum computing hardware is not only limited in the number of qubits available (defined formally in section 2.2) but is also prone to measurement noise.

Therefore, rigorous optimization of quantum programs is a necessity to obtaining the full value of the available hardware. Making exacting optimizations can be costly on classical computers, so we employ the quantum hardware itself to speed up this process.

**Prior Work** Numerous schemes for quantum program optimization have been proposed. Some optimization techniques assume certain various hardware schemes [2, 3] and make assumptions about the constraints on circuits [3,4]. Other systems' aims are to generate more efficient programs by making local optimizations [5], greedy optimization choices [6,7], and by employing heuristic search [7,8]. We propose an approach to quantum program optimization which generates highly optimized quantum programs for an arbitrary quantum computing hardware specification. This approach models the optimization of the quantum program as a constraint satisfaction problem. Critically, not only does this approach for optimizing quantum programs work on classical machines, but can be significantly accelerated with the use of a quantum computer itself. The proposed approach is highly extensible, configurable and tunable for the needs of various applications.

### 2.1.1 Motivating Quantum Program Optimization

It is not immediately obvious the advantages and necessity for optimizing quantum programs, but we argue that the cost reductions brought by a system such as the one we shall present will be critical to the future of efficient (and cost effective) quantum computation. The following are some issues with quantum computing that can be solved with optimization of the quantum programs without advancements in hardware technology.

**Time as a limited resource:** Intrinsic to the quantum mechanical effects that quantum computers operate using is the notion of *quantum decoherence*. Quantum decoherence is an effect caused by the environment trying to "measure" a quantum state before the actual quantum computer has a chance to measure it, destroying the computation result. This is an issue in quantum computing hardware, and is being addressed with error correcting qubits, and circuit optimization techniques. This process is probabilistic, and the longer our quantum program runs, the higher the chance that the computation result will be corrupted. A maximal run time for quantum programs for a given hardware is determined so that the error rate of computation is not greater than some threshold (i.e. 1/3). Due to this error rate, quantum computers sample measurements of many computations to approach on a correct result. Program optimization techniques can be employed to reduce circuit computation time, reducing result noise as well as transforming infeasibly long programs to ones that can be executed.

**Abstracting hardware architecture:** These optimizations also abstract away the programmers need for full and explicit knowledge of the quantum computing hardware. This makes programming for a quantum computer more accessible to those who may not need to know about the hardware. This abstraction allows quantum programs to be written as a virtual or logical design, which is later mapped to physical hardware by our system.

**Solving hard satisfiability problems:** Satisfiability (abbreviated SAT) is the problem of finding an assignment to a set of variables that makes a formula true (see subsection 2.2.4 for more information). Runtime for general Boolean SAT problems on classical computers runs in exponential time ($\mathcal{O}(2^n)$). On a quantum computer, we can get a quadratic speedup for

this by performing Grover's algorithm, arriving at a runtime of $\mathcal{O}(2^{n/2})$ [1]. If the satisfiability problem is formulated carefully, quantum backtracking algorithms can potentially provide an exponential decrease in runtime [9,10]. This can be used to solve larger problems in the same amount of time. We will employ these algorithms to make quantum programs efficient, for the given hardware. Using this scheme we can create a scalable quantum program optimization system that can generate optimal programs, outperforming other state-of-the-art systems.

**Self-hosting quantum circuit optimization:** Using quantum computers to synthesize optimal quantum programs is critical to increasing the complexity of quantum computations that can be performed. Quantum hardware will always constrain the capability of quantum programs, regardless of the technological advances made to quantum computing. By employing quantum satisfiability algorithms to find optimal quantum programs for an arbitrary hardware, we allow for more exacting optimizations to be applied within a reasonable time.

### 2.1.2 Significance of Self-Hosting

Self-hosting can be described as the property of a system which is capable of using it's own resources to perform or improve its function [1]. This concept can be expressed in many modalities from compilers to operating system kernels. For example, we could construct a C compiler in an assembly/machine language (called a bootstrapping compiler), then rewrite the C compiler in the C programming language and compile it. The result of this compilation would be a self-hosting C compiler. The following are several examples of self-hosting systems:

- C compiler (written in assembly) compiles a C compiler (written in C) which then compiles itself.
- Linux running a virtual machine of Linux.
- A Git repository containing the source code for Git.
- A processor running an emulation of itself.
- An optimizing C compiler optimizing itself.

Once a technology can be implemented such that it is capable of self-hosting, there can be many advantages. Self-hosting marks a milestone for the capabilities of a technology as well as acting as benchmark for the system itself. In our case, hosting the program optimization procedure on quantum hardware can potentially speed up the procedure.

[1] This is a looser definition than may be found in other texts. Others may state that self-hosting systems are strictly those that perform their function on themselves, but we relax this.

## 2.2 Concepts and Notation

It is necessary to introduce several concepts and some notation before the primary matter. The following will briefly introduce qubits, quantum gates and quantum programs. An introduction to basic notions of Boolean formulas and the decision problem of satisfying said formulas is also provided.

### 2.2.1 Qubits

Quantum bits, shortened to qubits, are the quantum equivalent of classical bits. A qubit takes on a quantum state, which can be represented mathematically by a tensor of dimension one with a size of two. Each entry of this quantum state vector is the complex value amplitude of the state being in the zero and one state, respectively. This is displayed formally in Figure 2.1. The number of qubits available in a quantum computer can be thought of as the memory resource that we are constrained by when writing quantum programs.

$$|q_i\rangle = \begin{bmatrix} a_0 \\ a_1 \end{bmatrix} = a_0 |0\rangle + a_1 |1\rangle \tag{2.1}$$

$$a_0, a_1 \in \mathbb{C} \wedge \sqrt{(a_0)^2 + (a_1)^2} = 1 \tag{2.2}$$

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{2.3}$$

Figure 2.1: The mathematical representation of a qubit. Note that $a_0$ and $a_1$ are the amplitude for the corresponding qubit's states.

We will use the notation $q_i$ for qubits in order to uniquely identify them. The quantum state of the qubit $q_i$ is represented by the tensor described above, which is sometimes called the wave function $\psi$. The special quantum states $|0\rangle$ and $|1\rangle$ represent the basis states which we call the *computational basis states*. These are named as such as we are usually only going to (or are able to) initialize computation of a quantum program with these bases.

Qubits are realized physically in various implementations, which we are not particularly concerned with in this report. A quantum register is a sequence of qubits in a specified order which can be represented by a joint state. The state vector representing the joint quantum state of a qubit register is computed by taking the tensor product of the individual states. The quantum register is shown mathematically in Figure 2.2.

Note though that once we apply an operation to the register basis, we may not necessarily be able to separate the register state vector into its components. When this inseparability occurs, we say that the quantum state or qubits in the register are *entangled*.

### 2.2.2 Quantum Gates

A quantum gate is an operation which acts on quantum registers. Quantum gates are represented mathematically by a unitary operation as shown in Figure 2.3. The gate operation

$$\mathcal{R} = [q_1, ..., q_n] \tag{2.4}$$

$$|\mathcal{R}\rangle = \begin{cases} (\forall q_i \in \mathcal{R})[|q_i\rangle \in \{|0\rangle, |1\rangle\}] & \bigotimes_{i=1}^{n} |q_i\rangle \\ \text{otherwise} & \begin{bmatrix} a_{00...0} & a_{00...1} & ... & a_{11...1} \end{bmatrix} \end{cases} \tag{2.5}$$

$$a_{00...0}, ..., a_{11...1} \in \mathbb{C} \wedge \sqrt{(a_{00...0})^2 + ... + (a_{11...1})^2} = 1 \tag{2.6}$$

Figure 2.2: The mathematical representation of a qubit register.

takes as input a qubit register (state vector) which it will act on. The operation output is the transformation described by the gate. Formally, this is the result of matrix multiplication of the state vector with gate unitary matrix.

$$U \in \mathbb{C}^{2^n} \times \mathbb{C}^{2^n} \tag{2.7}$$

$$U^\dagger U = I \tag{2.8}$$

Figure 2.3: The mathematical representation of a quantum gate's unitary operation $U$ on $n$ qubits. The dagger ($\dagger$) denotes the conjugate transpose of the matrix.

### 2.2.3 Quantum Programs

Quantum programs or quantum circuits are the construct used to compose quantum operations on individual qubits or qubit registers. There are two canonical, but equivalent, ways to represent quantum programs. The first is a textual, code-like, assembly language which gives a description of the quantum registers and the gates in the circuit in order they should be computed. The most common implementation of such a language is called the Open Quantum Assembly Language or OpenQASM [11]. The other way quantum programs are represented is by a visual approach. Gates are laid out as time goes on from left to right. An example of each of these representations can be seen in Figure 2.4.

Note that the quantum assembly code is a linearization (topological ordering) of the directed acyclic graph representing data dependencies between gates in the program.

### 2.2.4 Boolean Satisfiability

The Boolean satisfiability problem (frequently abbreviated as SAT) is a decision problem which decides whether there exists an assignment of truth values to Boolean variables which satisfies (makes true) the given Boolean formula.

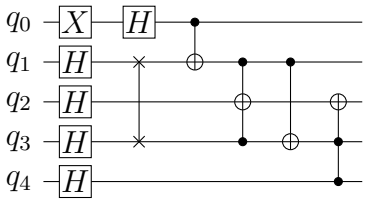
All Boolean formulas can be converted to a logically equivalent canonical form called conjunctive normal form (CNF). Conjunctive normal form represents a Boolean formula as a conjunction (AND operation) of disjunctions or clauses (OR operations). In each clause,

```
OPENQASM 2.0;
include "qelib1.inc";

qreg q[5];

x q[0];
h q[0], q[1], q[2], q[3], q[4];
swap q[1], q[3];
cx q[0], q[1];
ccx q[1], q[3], q[2];
cx q[1], q[3];
ccx q[3], q[4], q[2];
```

(a) A simple quantum program written to the OpenQASM standard. Each line of the program shows the operation to be performed followed by the qubits it operates on.




(b) A simple quantum circuit. This is how quantum programs are typically described diagrammatically. Each line represents a qubit and read from left to right shows how gates are applied to qubits at each time step.

Figure 2.4: Two methods of representing the same quantum program. One shows an assembly-like representation in Open QASM and the other is a graphical circuit-like diagram.

the variable and the existence or non-existence of a negation are referred to together as a literal. This representation is the most common form used in solving SAT problems. An example of a Boolean formula and its CNF representation can be found in Figure 2.5.

The SAT problem belongs to the complexity class of NP-complete problems [12], making it a challenging problem to solve for arbitrary Boolean formulas. However, state-of-the-art SAT solvers which have been developed over the past two decades are very efficient with real world problem instances [13]. Formulating problems in the language of a constraint satisfaction problem lends itself to be converted to and solved as a SAT problem. This work concerns itself with constraint satisfaction to solve the quantum program optimizations, and transform our problems to SAT prior to solving as part of the optimization process.

$$\textit{Boolean Formula} \qquad (x_0 \to x_1) \wedge (x_2 \vee x_1) \tag{2.9}$$

$$\textit{CNF} \qquad (\neg x_0 \vee x_1) \wedge (x_2 \vee x_1) \tag{2.10}$$

$$\textit{Satisfying Assignment} \qquad x_0 = F, x_1 = T, x_2 = F \tag{2.11}$$

Figure 2.5: A boolean formula and its corresponding CNF formula. A satisfying assignment for both equivalent formulas is below both.

# Chapter 3

# Background

To the best of our knowledge, no evidence of prior work on quantum program optimizations using quantum computers has been identified. Prior work has been conducted in the areas of quantum program optimization and quantum search algorithms. A search into these topics was conducted in order to bridge these technologies into the desired compilation system. The following sections describe the landscape of classical quantum program optimization schemes and quantum satisfiability algorithms.

## 3.1 Quantum Program Optimization

Quantum program optimization has been approached in a multitude of ways. using templates to aid in simplification of quantum program [5], constraint based optimization [2], temporal planner based compilers [14], as well as other tactics. These methods are suitable for very specific use cases but garner issues around generality, optimality and scalability. The following summaries describe just a few of these methods.

### 3.1.1 Template-Based Simplification

The quantum program optimization tactics proposed by Maslov et al. [5] work by making local optimizations via a heuristic approach. In a fashion similar to peephole optimization, the proposed system uses templates or patterns within the quantum program and swaps them with simpler and faster equivalent operations. This has the result of reducing the overall runtime of the program, which can have impacts on the result accuracy when the program is run, and can make long programs short enough to be executable. Even though the programs this system can work on are restricted to only using certain operations, more work could be done to alleviate this restriction. Employing such a system may be useful for preprocessing circuits prior to a more rigorous optimization procedure.

### 3.1.2 Constraint Based

A constraint based approach to quantum circuit optimization has been proposed for noisy intermediate scale quantum computers by Murali et al. [2]. This approach, although similar

to our constraint based model, assumes a specific hardware layout of the qubits similar to the layout provided in IBM's 16 qubit quantum computer. Their method takes inspiration from path routing rules used in circuit board design, mapping swap operations onto the mesh layout of the qubits. Although this work is efficient for such architectures, our proposed approach aims to be general and configurable for any hardware, without sacrificing performance. Other constraint based approaches have been proposed which follow similarly by framing the task of optimizing or compiling quantum circuits to a given hardware [4, 15].

### 3.1.3 Minimum Linear Arrangements

Pedram and Shafaei provide a system for minimizing the number of SWAP operations needed to allow two-qubit operations to be performed [3]. In their model, they only consider hardware which is arranged in a linear nearest neighbor architecture. They construct a graph of the qubits which interact through two qubit operations and construct a minimum linear arrangement problem which they solve and use that to assist in a greedy, backtracking-like mechanism to schedule gates into a solution.

### 3.1.4 Gate Transformation and Commutation

Gate transformations and commutation can be performed to make quantum programs more resource efficient by exchanging and swapping more efficient gates or sets of gates for less efficient ones. In this method [7], they propose such a model which allows gates to be reformulated from time inefficient gates to ones which utilize as many qubits as possible at each time-step. This approach is similar in principle to the template based approach but varies in that it allows for a more general and granular notion of equivalent substitutes for gates.

### 3.1.5 Temporal Planners

Another method to mitigate quantum state decoherence and reduce runtime of quantum programs is by the use of a temporal planner. State-of-the-art temporal planners have been assessed for completing this task for a particular quantum algorithm on linear nearest neighbor hardware arrangements [14]. Although a general approach, it seems to be similar to the constraint based systems in both design and goal.

## 3.2 Quantum Satisfiability Algorithms

In order to self-host the program optimization procedure on a quantum computer, this architecture uses existing quantum search algorithms. Part of the optimization procedure will be to check if there exists a valid transformation of the circuit given some desired properties of that circuit. Therefore we need a quantum algorithm to find if such a solution exists. The following algorithm are several general approaches for exactly this process.

### 3.2.1 Grover's Algorithm

Grover's algorithm is a quantum algorithm that performs search of an unordered database faster than classical algorithms [1]. The classical algorithm for this problem runs in linear time ($\mathcal{O}(n)$). This database within Grover's algorithm is represented formally as an oracle function. For our case, the oracle is implemented as a Boolean formula testing the satisfiability of some set of Boolean logical clauses. Another way to describe this algorithm is that it returns the unique assignment of values to variables in a function which produce a given output. It cannot conclude the non-existence of such an assignment, only the existence of such a solution. For Boolean functions, Grover's algorithm can find a satisfying assignment to the function, if one exists, but cannot state whether it does not exist. Grover's algorithm runs in $\mathcal{O}(\sqrt{n})$, and is optimal for this task. This quadratic speedup over the classical algorithm inspires the self-hosting optimization we wish to perform.

Although Grover's algorithm offers quadratic speedup, trivial instances of our circuit optimization model would require of Grover's algorithm a number of qubits which would be unrealistic to realize on actual hardware, at least as of the time of writing. This is because a logical formula in conjunctive normal form which we want to represent in a quantum program requires one qubit for each Boolean variable in the formula, one qubit for each clause, and one qubit for the disjunction of all of the clauses. Our model will produce such formulas with possibly several hundred Boolean variables on trivial input circuits.

There are some techniques to mitigate this which have been explored. Cerf and Grover give a version of the original Grover's algorithm algorithm which allows for nested runs of Grover's algorithm [16]. A block based approach to Grover's algorithm checks for existence of a solution given a partial assignment to the variables [17]. Both of these techniques should be examined further by those wishing to exploit the structure of search problems. The technique we employ takes inspiration from Dunkjo [18] where they propose a hybrid quantum-classical solver which performs backtracking classically until the solution can be simplified to be small enough to fit an Grover's algorithm circuit on a small quantum hardware. This is discussed in detail in chapter 5.

Although Grover's algorithm is less complex to implement than others, there are other approaches. We explore this next approach, quantum backtracking, as an alternative.

### 3.2.2 Quantum Backtracking

Backtracking is the task of incrementally constructing a partial solution to a constraint satisfaction problem until either the partial solution is found to be invalid (in which case we revert the incremental change), or a complete and valid solution is found. Many classical algorithms exist to perform backtracking, but we are interested if there are quantum backtracking algorithms, and if so, whether they outperform classical algorithms.

It has been shown that, in theory, a quantum backtracking algorithm can outperform the equivalent classical algorithm [19, 20]. Montanaro realizes this in an approach to quantum backtracking algorithms via quantum walks [9]. This algorithm, under certain careful circumstances, can have an exponential reduction in runtime as compared to an equivalent classical backtracking algorithm. A study of the efficiency of this quantum algorithm concluded that random instances can still realize speedup factors on the scale of orders of

magnitude over classical counterparts [21].

Several improvements to both the performance [10] and resource usage [22] of this algorithm have also been formulated. Performance improvements were made which improved the worst case runtime complexity of the algorithm. The original work's worst case runtime depends on the size of the backtracking tree of assignments to Boolean variables [9] whereas this method only depends on the size of the tree which is explored during search. Under practical conditions, it is expected that this tree is is significantly smaller than the full backtracking tree produced by [10]. Resource usage was reduced both for the number of data qubits required as well as for circuit runtime [22]. An analysis of methods for ordering of exploration of the variables was performed. The original algorithm was also iterated upon by Montanaro to perform branch-and-bound tasks [23].

### 3.2.3 Quantum Approximate Optimization

The Quantum Approximate Optimization Algorithm (QAOA) is a quantum approximate combinatorial optimization algorithm [24]. Combinatorial optimization is the task of finding an assignment to variables which satisfies the maximum number of assertions from a corpus. Approximate combinatorial optimization guarantees that we find a solution which generates solutions that approximate the maximum number of satisfied assertions to within some ratio $\epsilon > 0$. Such algorithms could be used to find approximate maximally satisfying assignments, which would be a useful mechanism for this work. This was not pursued as it is not well understood if QAOA can produce similar quality approximately optimal solutions with less time than classical approximate optimization algorithms.

### 3.2.4 Quantum Genetic Algorithms

Genetic algorithms are a class of approximate combinatorial optimization algorithms which borrow notions from biology to *evolve* solutions. Generations of potential solutions are considered *chromosomes* and have attributes iteratively mutated, and solution crossover is applied between pairs as is done in real organism populations. Quantum genetic algorithms are the quantum computing analog of genetic algorithms. Work has been done to implement quantum genetic algorithms [25, 26]. Although it does seem that there are statistical advantages to using quantum genetic algorithms over their classical counterparts, it suffers the same issue that QAOA has: it is not clear yet if there exists a performance improvement over classical methods for the same problem.

# Chapter 4

# Quantum Program Optimization

## 4.1 Introduction

This chapter will discuss the architecture for optimization of quantum circuits. There are a series of phases to the proposed system, which are outlined in the following section (section 4.2). In order to iteratively arrive at an optimal quantum program, a control procedure is implemented to handle this process. It is provided a program transformed according to the constraint model, and arrives at the optimal circuit. This process is outline in section 4.3.

## 4.2 Procedure for Self-Hosted Quantum Optimizations

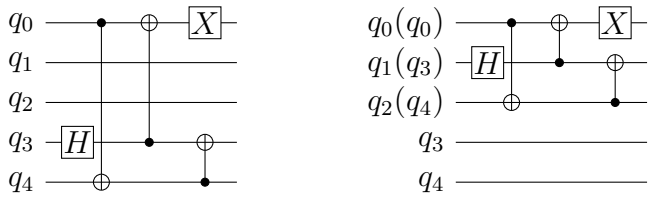
The following is an outline of the end-to-end procedure used to optimize quantum programs using a self-hosting optimization approach.

1. **Input Quantum Circuit:** We take as input a virtual quantum circuit which we want to optimize. A virtual quantum circuit is one that does not have any mapping of the circuit to physical hardware. The quantum registers within the circuit can then be thought of as virtual quantum registers.

2. **Quantum Circuit Preprocessing:** Minor preprocessing must be performed on the input circuit to ensure it will be compatible with the given hardware. This includes checking that the circuit qubit register is not bigger than the number of available qubits on the machine. If the given hardware does not support a certain gate in our input circuit, an automated process should try to find a reasonable decomposition of this gate into several compatible gates.

3. **Pack Qubits:** Some quantum circuits our system will receive will not be *packed*. A quantum circuit is packed if it is as a virtual circuit whose only utilized qubits are consecutive, as well as lexicographically minimal. This means that there are no qubits unused (i.e. no gates on the qubit) within the circuit between used qubits that do have gates on them. This also implies that the qubits of least index should be used first. This is shown visually in Figure 4.1. As part of our preprocessing procedure, we remap qubits to be consecutive with each other and expand the circuit register to be

equal to the number of qubits available to the quantum hardware. Taking the original circuit as input to the CSP generation process would add unnecessary complexity, so we re-order the qubits in this manner to simplify the next steps. We hold onto the mapping between unpacked and packed circuit so that we can reverse this mapping later on.




(a) An unpacked circuit. The qubits being used are $q_0$, $q_3$, $q_4$. We prefer to take in a circuit with the lexicographical ordering of the qubits.

(b) A packed, consecutively ordered remapping of the circuit in Figure 4.1a. The qubits being used are $q_0$, $q_1$, $q_2$.

Figure 4.1: An example of the packing process of virtual quantum circuits, before and after.

4. **Generate Constraint Satisfaction Problem from Input Circuit:** The packed circuit is used in a process which generates the constraint satisfaction problem (CSP) based on the model outlined in chapter 6. This process generates a quantifier free finite domain (QFFD) constraint satisfaction problem. To be a QFFD CSP, a problem's constraints must be expressible without using logical quantifiers and all variables must be bounded by constraints such that they can only take on a finite set of values. The quantum algorithms that will solve the CSP will expect all variables to be of a Boolean type. There is a procedure which we can use to convert a QFFD problem to a Boolean QFFD problem by generating new variables which correspond to the setting of an integer variable to one of the finite set of values which it can take on. The mappings between the original variables and their new Boolean counterparts is maintained so that we can reverse this process. At this point, if we wish to run the optimization on a classical solver such as Z3 [13], we could use the CSP at this point for this task. We regularly do this for validation purposes, using small problem instances.

5. **Transform CSP to Boolean CNF:** The next phase performs an automated transformation process from a Boolean QFFD problem to a Boolean formula in conjunctive normal form (CNF). Logical operators which are not conjunction and disjunction need to be removed, for techniques for this are well known. The method for this, given by Tseitin [27], introduces new variables to obtain a CNF formula linearly increasing the size of the formula. There are some constraints which span many variables, so once this transformation is complete a procedure splits the clauses which are longer than some given length into pieces. Keeping these clauses at their normal length is unwanted as the required gate for that clause would have to be decomposed into too many sub-operations. When splitting the clause into smaller parts, an auxiliary Boolean variable

is added to each of the clauses to tie the two clauses together, mimicking the same truth values as the original clause.

6. **Start Optimization Controller:** The optimizer is started and passed the hardware description, and the constraint satisfaction problem in its present CNF form. This controller runs until the an optimal solution to the problem is found. It returns the complete assignment of values to the CSP variables. Those values are then used in construction of an optimized physical circuit. The process of this optimization is outlined in section 4.3.

7. **Construct Optimized Physical Circuit from Model:** The optimized physical circuit is constructed from the assigned values returned by the optimizer. The physical circuit, is a quantum circuit which is mapped to hardware. We use the original circuit along with the physical-to-virtual qubit mappings and SWAP gate insertion variables to construct the physical circuit. The qubit mappings instruct on how the circuit input and output should be mapped when the new circuit is ran and measured. Gate qubit mappings are used to remap gate registers. Assignment to SWAP insertion variables will inform the placement of new SWAP gates. These are the items which are necessary to building the physical circuit. The descriptions of these constraint variables are covered in detail in section 6.4.

## 4.3 Controller for Optimization Solver

This controller's goal is to delegate execution of quantum SAT solver programs to the quantum computer hardware, until the optimal solution is found. This optimization scheme requires that we have a process which controls the optimization procedure using a classical computer to perform the delegation to a quantum computer. The solver program receives as input an instance of the CSP modeled on the virtual quantum circuit as well as a description of the quantum computing hardware.

Once the constraint satisfaction problem is generated, this CSP is passed to the optimizer. This optimizer controls successive runs of a quantum satisfiability algorithm on quantum computer. In the following process, we are not concerned with the specific changes need to use one quantum satisfiability algorithm or another. Instead, the procedure is outlined with the intent that any suitable algorithm can be dropped in place. Note that the optimizer's goal is to find the most efficient circuit based on several optimization criteria. For each step the optimizer takes we will do the following:

1. **Decide Values for Optimization Criteria:** The optimizer will begin each step by determining the values for the optimization criteria which will be taken on by the CSP. The process for deciding these values is discussed in subsection 4.3.1.

2. **Fixing Optimization Criteria:** The provided CSP will have several variables for the optimization criteria, which constrain the other variables in the CSP. The optimizer will fix the optimization criteria to a specific value, decided by the prior step. For example, the CSP will have a variable for the optimization criterion specifying the

number of SWAP gates that can be inserted. This variable conditions other variables in the CSP.

The optimization controller will pick values for each optimization criterion in this manner and sets a strict equality constraint for each criterion. With each of these variables constrained in this manner, the solver will search for a solution to the problem given desired properties of the output circuit.

3. **Simplification of the CSP:** Now that constraints are set on the optimization criteria the controller will apply some fast (classical) simplifications to the CSP. This simplification step can be performed on the CSP at any time but is at least done at this point to propagate the fixed criteria variables as constants throughout the CSP. This has the potential to quickly eliminate many constraints or fix other variables, or at least possibly tighten their bounds.

4. **Construct Boolean Logic Oracle from CNF:** An automated transformation is then used to construct a quantum circuit to represent the Boolean logical circuit from the CNF formula. Depending on the quantum solver algorithm, this circuit is constructed differently. For Grover's algorithm, there exists a straight forward procedure [28] for constructing a oracle from CNF formulas. For quantum backtracking, techniques exist to construct clause checking circuits [9, 22].

5. **Quantum Solver Setup with Oracle:** We pass the oracle to a routine to generate the quantum solver circuit to be run on a quantum computer. This routine will fill in the solver circuit with the oracle where it is needed.

6. **Run Quantum Solver and Read Result:** The quantum solver circuit will then be dispatched to the quantum computer. A result will be obtained back from the solver circuit's measurement. Depending on the type of quantum solver algorithm being used, this step of the procedure could vary. This implementation specific routine could involve different control behavior depending on the solver algorithm chosen. In our case, this control process runs a hybrid quantum-classical solver. It uses a classical solver along with iterations of Grover's algorithm to build a satisfying assignment to the CSP.

7. **Use Solver Results to Inform Optimization:** The result of the solver will inform the setting of the criteria. The solver algorithm will return whether the settings of the criteria was feasible or not. If it was feasible, the assignment of values to variables is returned from this procedure in order to reconstruct the optimized physical circuit.

The optimization controller runs this process for each step until a stopping condition is reached. This stopping condition can occur once we have found the optimal result, or if no feasible result could be found. The final optimized assignment from the controller is passed to a set of routines to map the solver result output to the CSP variables and then to an optimal quantum circuit.

### 4.3.1 Objective Optimization

As part of the optimization procedure, values for the optimization criteria must be set at each step. There are multiple criteria which are used as objectives in the proposed constraint model. A Pareto optimization method is used to arrive at the best circuit possible under the model. This Pareto optimization function tries to achieve a *Pareto efficient* assignment to the criteria. A Pareto efficient result, in our case, is one in which we cannot minimize (and therefore improve) any of the criteria further without increasing another, or that further minimization would arrive at an infeasible result. The definitions of the criteria are outlined in section 6.6.

## 4.4 Conversion from Optimizer Solution to Optimized Circuit

After optimization is complete, that system will return a Boolean assignment to the SAT instance. We also have the original circuit and the mapping of CSP variables to Boolean variables. With all of this information we can convert the SAT assignment back into a quantum program. In our case the solver we use handles this on its own by holding onto the mappings of the CSP variables to the Boolean SAT variables and reverses this mapping given an assignment. CSP variables can be Boolean or integer variables and the latter are mapped to a binary representation.

# Chapter 5

# Quantum Program Solver

## 5.1 Introduction

The quantum satisfiability solver is the system in this architecture which takes an instance of the constraints on a quantum program, performs a quantum algorithm to search for a solution to satisfy the constraints, and outputs a solution which satisfies the constraints. It will also specify if no solution were found. In general, a satisfiability solver is an algorithm or computer program that decides on a solution to decision problem, such as a constraint satisfaction problem, and terminates by outputting a solution or stating the unsatisfiability of the problem. More specifically, we are concerned with Boolean satisfiability solvers, which operate on Boolean variables and formulas.

A quantum satisfiability solver embodies the same concepts as classical solvers, but use quantum algorithms to achieve the same goal. For both classical and quantum solvers there are several families of algorithms for solving satisfiability problems, but several quantum algorithms for this have been shown to asymptotically outperform their classical analogues [1, 9, 10].

In this system, the implementation consists of a hybrid quantum-classical satisfiability solver. This hybrid solver utilizes a classical solver to driver and hands off sufficiently small sub-problems to the quantum computer for solving. The specifics of this solver, and motivations for this choice are discussed in the following sections (section 5.2, section 5.3 respectively). Implementation specific details are discussed in section 5.4. We finish this chapter with a discussion of improvements and other solver schemes which could be in-place replaced with the hybrid approach.

## 5.2 Limitations

The space-complexity requirements of SAT will be at least proportional to the number of variables in the SAT problem, $\mathcal{O}(n)$. This is because the assignment returned will be of size $\mathcal{O}(n)$ bits. Therefore, as the assignment is constructed that $\mathcal{O}(n)$ bits must be available to store the assignment for the lifetime of the solver. The size of the Boolean SAT instances produced by our model cannot be solved in whole by a quantum algorithm for this type of search. Such an algorithm would require a qubit register of size at least $\mathcal{O}(n)$ to store

the final assignment. For Boolean SAT problems with more than around 70 variables, there does not yet exist a massive enough qubit register to support even the assignment for such problems, let alone state for the solver. Conversely, it is important to note that if we provide a quantum hardware with a number of qubits, say $\mathcal{O}(m)$, then problems of this size or smaller can be solved on the hardware.

In Vedran and Dunjko's work describing a SAT algorithm for small quantum devices, they discuss an approach obtaining speedup via a hybrid quantum-classical solver [18]. Given the $n$ Boolean variables solved by classical solver and $m$ is the number of qubits in the small quantum device. This systems uses a classical satisfiability algorithm until the assignment is of size $n - m$. As soon as the assignment is of this size, the remaining $m$ variables are solved by Grover's algorithm. The worst case time-complexity of a classical SAT solver is $\mathcal{O}(2^n)$ and Grover's algorithm runs in time $\mathcal{O}(2^{n/2})$. Therefore, it is stated that the worst case runtime of this hybrid solution would be:

$$\mathcal{O}(2^{n-m} + 2^{m/2}) \quad n \gg m \tag{5.1}$$

We can also consider $n$ as the number of bits used by classical solver and $m$ as the number of qubits. A complete assignment contains $n+m$ Boolean values mapped to Boolean variables. Naively, this new hybrid classical-quantum method would seem to be the superior algorithm against the purely classical approach for any setting of $m \in [1, n]$, but the authors ask us to consider average case performance of a simple SAT algorithm: Schöning's algorithm [29, 30]. Although by far not the current state-of-the-art, this algorithm has average case runtime complexity of $\mathcal{O}\left(\frac{4}{3}^n\right)$.

Let us consider now what settings for $n$ and $m$ that make the hybrid approach better than Schöning's algorithm:

$$2^{n-m} + 2^{m/2} > \left(\frac{4}{3}\right)^n \tag{5.2}$$

In order to do better than Schöning's algorithm for sufficiently large $n$ we must have some minimum ratio of classical bits to qubits representing the SAT problem which would make this hybrid approach better than Schöning's algorithm. Vedran and Dunjko find this ratio to be:

$$\frac{m}{n} > 0.74 \quad n \to \infty \tag{5.3}$$

This ratio is not favorable if we wish to employ a small quantum device such as those available at the time of writing for large SAT problems. Nonetheless, such a solver would prove that implementing a hybrid quantum-classical SAT solver is possible.

Although this analysis shows we may not do better trying to solve larger SAT problems with this approach, it has proved to be the simplest to implement. This is the way we have chosen to approach the limitations imposed by the state of quantum computing hardware while still solving larger SAT problems. [1]

[1] This analysis is a necessary reconstruction of that provided in [18]. Refer for more details in context as they apply the analysis differently as a pretext for wholly different approach from ours. The main difference is that they go on to propose a method which does not suffer from a "threshold" of qubits to classical bits at which the polynomial-time speedup is realized.

## 5.3 Hybrid Solver

The hybrid classical-quantum solver copes with the aforementioned limitations by making use of a classical backtracking SAT solver to arrive at partial solutions which can allow the quantum SAT solver to manage small problems. Traditional backtracking SAT solvers examine partial assignments to variables in a SAT instance by adding literals to the assignment recursively. At each recursive step, all of the clauses must either be satisfied or have an undefined truth value. If the assignment derives a clause which is false, we remove the latest literal in the assignment (we "backtrack") and try an alternative path to a satisfying assignment. In our approach we take advantage of the varying length of these partial assignments at various points in the backtracking procedure.

We modify the classical backtracking technique to accommodate the quantum solver such that it respects the quantum hardware's memory resources. By exploiting the recursive structure of the backtracking procedure we can make additional checks at each step of the this process for our purposes. In this case, each recursive step is augmented with an additional check to learn if it is possible to run the quantum solver algorithm at this point in the backtracking algorithm.

The number of qubits used by the quantum SAT algorithm will be directly related to the number of unassigned variables and clauses not yet satisfied. Given a partial assignment and the clauses for the problem, the solver applies unit propagation of only the assignment literals to the clauses to eliminate all of the clauses satisfied by the assignment. A unit is a clause which contains one literal. Unit propagation is a process which allows us to remove literals from clauses as well as entire clauses from a CNF formula which contain a clause which is a unit. All other literals are left as they are. This now simplified problem is much smaller now that it does not contain the unnecessary information of satisfied clauses and any instances of the assigned variables.

The number of unassigned variables and clauses can now be calculated and then passed to a mathematical expression which describes the number of qubits used by the chosen quantum algorithm. In this implementation the whole expression is represented as:

$$\begin{aligned} \#\ \textit{of Hardware Qubits} \geq 1 + \#\ \textit{of Unassigned Variables} \\ + \#\ \textit{of Clauses without Truth Value} \end{aligned} \tag{5.4}$$

We are using Grover's algorithm with a basic layout for the quantum SAT solving procedure but note that this expression would differ depending on the quantum SAT method that is used. The expression describes parametrically that the quantum procedure requires a qubit for each variable which it will solve for as well as each clause, plus one more for the indicating if the sub-problem is satisfied. If the resulting number is greater than the available qubits on hardware the classical backtracking will be informed to continue the classical process. If in fact the number of qubits needed is less than or equal to the number of available hardware qubits we can attempt to use the quantum solver to complete the problem.

At this point the quantum solver procedure must build a quantum program to run on a given quantum computer. This procedure is given the simplified set of clauses and will return one of the following:

- A satisfied result containing the assignment of truth values to each variable.
- An inconclusive result for if a satisfying completion of the assignment was not found.
- An unsatisfied result to tell the solver that it found proof of unsatisfiability.

If the satisfied result is received it is applied to the classical assignment and the solver return the assignment in full to the variable remapping and optimization procedures. Otherwise with an unsatisfied result the solver will be informed the sub-problem is unsatisfied. Lastly, an inconclusive response prompts the solver to continue as if the quantum solver was never run.

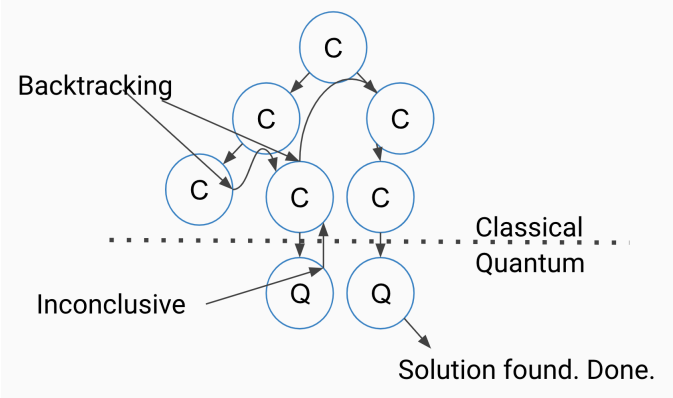



Figure 5.1: An example of the backtracking "tree" where each node is a decision or some set of decisions being made on a Boolean variable or variables. The "C" nodes are those performed by the classical solver and "Q" are nodes solved by the quantum solver. Note that backtracking will go back up the tree and to some new truth assignment for a decision above the failed node.

## 5.4 Implementation

Our hybrid SAT solver implementation uses the Z3 satisfiability modulo theory (SMT) solver [13] and the QuEST [31] quantum simulator.

The Z3 SMT solver contains a state-of-the-art SAT solver. Z3's SAT solver performs backtracking along with myriad improvements on the baseline solver. Z3 also has tactics for converting problems from QFFD problems to Boolean SAT, which is used by our system prior to arriving at the solver. This SAT solver was modified to execute two callback functions for the purpose of checking the qubit requirements and running the solver respectively. Our system allows the user to change these callbacks so that a new check can be easily replaced for a different quantum algorithm, hardware configuration or so that the simulated solver can be easily replaced with a call to a quantum computer.

A quantum simulator is used and not actual quantum computing hardware due to the limitations of current quantum hardware. It was determined that the most fit-for-purpose

quantum computing hardware available comes from IBM's Quantum Experience, 15-qubit machine but it was inadequate nonetheless. The length of circuits we would have needed to run would have exceeded the capability of this system. Also, the required number of requests to their system for one run of this whole system would exceed the feasible throughput of requests to this hardware. Due to this, the alternative has been to perform simulation of the quantum circuits which otherwise would be run on quantum hardware. The simulator chosen was the Quantum Exact Simulation Toolkit (QuEST) for its performance and scalability to distributed and GPU systems [31].

Grover's algorithm was implemented within the QuEST framework so that it could receive CNF Boolean formulas and convert them to a quantum oracle. The original formulation of Grover's algorithm assumes a unique solution satisfying the function or oracle. The algorithm can be extended to support search for any solution of a known number of solutions $k$ by reducing the number of iterations which the algorithm runs for by said factor $k$. Further, if the number of solutions is not known, then Grover's algorithm should be run multiple times, incrementally increasing the number of iterations with each run. At the end of each incremental run the probabilities of measuring each assignment are computed and the assignment which is most probable is checked against the original clauses for satisfiability. If that solution does not satisfy all of the clauses, Grover's algorithm is rerun with the next number of iterations. If satisfied, the quantum solver is done and returns the result to Z3. A maximum number of iterations is set and if a solution is not found within that number of iterations, the result returned to Z3 is that the QuEST solver was inconclusive in its finding of a solution. Therefore, Z3 will be informed that it should continue as if the quantum solver was not run.

Once a satisfiable result or no solution is found, Z3 returns the assignment of Boolean variables. This is then returned to the optimization process for analysis of the optimization criteria and conversion back to a quantum circuit.

# Chapter 6

# Quantum Program Constraint Model

## 6.1 Introduction

The following chapter discusses the details of the constraint model used in the optimization process. The constraint model requires that a virtual quantum circuit, a target quantum computer hardware description and settings for the optimization criteria be provided in order to generate an instance of the model. Assignments to the constraint variables will inform the optimization process as to how to transform the original circuit into an optimized, physically-mapped circuit according to the model.

The following section (section 6.2) will discuss the problem addressed by this model in detail. The next sections describe the model as a formal constraint satisfaction problem. The first section (section 6.3) describes and instruments major constructs within the model, such as quantum circuits, devices and gates. A section devoted to the variables used in the model (section 6.4) as well as the constraints on those variables (section 6.5) is provided. Lastly, a section about the model's optimization criteria (section 6.6) and how they are calculated using constraints, is provided.

## 6.2 Problem Statement

This constraint model aims to yield solutions to the following problem: Given a virtual quantum circuit, $\mathcal{C}_{\text{virt}}$, (i.e. where qubit registers are virtual or logical) and a hardware description for a quantum device, $\mathcal{M}$, compute the necessary details needed to build an optimized physical circuit, $\mathcal{C}_{\text{phys}}$, under the constraints of the hardware and criteria. The general goal of the optimization criteria are to accomplish the following:

- **Minimal Circuit Width:** As a preprocessing step, the circuit is packed so that the physical number of qubits ever touched by the realized quantum circuit is minimized.
- **Minimal Circuit Depth:** We want to minimize depth to allow us to run circuits which would be otherwise unrealizable. In our current scheme, circuit depth is the same as circuit end time.

- **Minimal Total Gate Count:** This is the total number of gates in the circuit. This criterion is a good approximate for the depth measure. In our case, we can just measure the number of SWAP gates added, as the model does not account for any gate elimination or substitution yet.
- **Minimal Noise:** We need to minimize the measurement noise by using and as few as possible of the noisiest gates that some hardware implementations provide.

Values for an absolute maximum circuit depth ($d_{\max}$) and gate count ($g_{\max}$) must also be defined. As of now, the equivalent value to maximum circuit depth is maximum circuit time ($t_{\max}$). The section on timing (subsection 6.3.4) gives insight as to why the terms are equivalent for now. These values are provided as a technical measure so as to keep the objectives within a finite domain.

In order to generate an instance of this model, an instance of the constructs formalized in section 6.3 must be provided. In particular one quantum circuit and one quantum device must be defined. Also necessary are settings for the maximum bounds on certain variables as outlined. Settings for optimization criteria must also be given. Given all of these settings, a well defined instance of the model can be formulated.

Note that gate depth and gate count are, in some cases, affected by the other and in other cases are mutually exclusive. Also of note is that this model is not proven to produce the perfect solutions yet, nor is this claimed. For an example of a case where this model would not be able to improve the input circuit, see section 8.3, which gives an example of such a situation.

## 6.3 Constructs

There are several overarching model constructs that are central to formulating this constraint model. In this section, formal descriptions of these constructs are provided.

### 6.3.1 Quantum Circuit

A quantum circuit is the description of the ordered set of operations (quantum gates) to be applied to the qubit register. In order to construct an instance of this model, one such circuit must be provided. We define a quantum circuit formally as:

$$\mathcal{C} = \langle \Gamma, \mathcal{R}_{\text{virt}} \rangle \tag{6.1}$$

$$\Gamma = (\mathcal{G}_i | \mathcal{G}_i.\mathcal{R}_{\text{gate}} \subset \mathcal{R}_{\text{virt}} \wedge i \in [1..g]) \tag{6.2}$$

$$\mathcal{R}_{\text{virt}} = \{q_1, ..., q_m\} \tag{6.3}$$

Where the quantum circuit, $\mathcal{C}$, is defined by a tuple containing the following:

- $\Gamma$: The sequence of $g$ gates in the order they are applied to the circuit qubit register. Each gate must use qubits from the circuit register.

- $\mathcal{R}_{\text{virt}}$: The virtual qubit register of the circuit, which is a set of $m$ virtual qubits. A virtual qubit, sometimes called a logical qubit [7] are not assigned to actual hardware qubits in a one-to-one mapping. They are therefore virtual or logical qubits as they do not have a physical mapping.

### 6.3.2 Quantum Gate

Quantum gates are the operations that act on qubits. The operation that is performed on a quantum state is called a *unitary operation*. A unitary operation can be represented by a square matrix whose conjugate transpose is the operation's own inverse. We define a quantum gate as:

$$\mathcal{G} = \langle \mathcal{R}_{\text{gate}}, U_{\text{gate}} \rangle \tag{6.4}$$
$$\mathcal{R}_{\text{gate}} = \{q_1, ..., q_r\} \tag{6.5}$$
$$U_{\text{gate}} \in \{U : U \in \mathbb{C}^{2^r} \times \mathbb{C}^{2^r} \wedge U^\dagger U = I\} \tag{6.6}$$

Where the quantum gate, $\mathcal{G}$, is defined by a tuple containing the following:

- $\mathcal{R}_{\text{gate}}$: The gate input and output qubit register within a circuit. This register is of size $r$.
- $U_{\text{gate}}$: The unitary matrix representing the operation performed by this gate. All gates must be unitary except those that perform measurement.

### 6.3.3 Quantum Device

A quantum device is a real world machine which allows us to run quantum programs. It describes the gates and registers allowed by circuits executable on the machine. It also establishes where two qubit gates are allowed between qubits. We define a quantum device as:

$$\mathcal{M} = \langle \mathcal{R}_{\text{phys}}, C, \Sigma_{\mathcal{G}}, t_{max} \rangle \tag{6.7}$$
$$\mathcal{R}_{\text{phys}} = \{q_1, ..., q_n\} \tag{6.8}$$
$$C = \{(q_u, q_v) : q_u, q_y \in \mathcal{R}_{\text{phys}} \wedge q_u \neq q_v\} \tag{6.9}$$
$$\Sigma_{\mathcal{G}} \subseteq \mathcal{U}(\mathcal{G}) \tag{6.10}$$

Where the quantum device, $\mathcal{M}$, is defined by a tuple containing the following:

- $\mathcal{R}_{\text{phys}}$: The physical qubit register of the device, which is the set of $n$ physical qubits. A physical qubit has a one-to-one correspondence to a specific qubit within some real-world device.
- $C$: The physical hardware qubit coupling map. The coupling map describes which qubits are allowed to have two qubit gates between them. It is represented as a directed graph. An edge from $q_u$ to $q_v$ indicates that a two qubit gate can be placed between them with the gate control at $q_u$ and gate target at $q_v$. A example of such a graph is shown in Figure 6.1.

- $\Sigma_{\mathcal{G}}$: The alphabet of quantum gates that can be represented on this hardware. It is a subset of all possible gates in the universe of gates, $\mathcal{U}(\mathcal{G})$. Usually we will have some shorthand to denote common gate types (for example, X for Pauli X, H for Hadamard, CNOT for conditional negation, and SWAP for the swap gate).

- $t_{max}$: The maximum time the device can run a circuit for in discrete time steps. This is a simplification of reality for now, which eliminates circuits which are too long for the device.

As in automata theory, we can say that there is a language of quantum circuits, $L_{\mathcal{C}}$, compatible with some quantum device $M$. This language can be thought of as a subset of the universe of all possible quantum circuits:

$$L_{\mathcal{C}}(\mathcal{M}) \subseteq \mathcal{U}(\mathcal{C}) \tag{6.11}$$

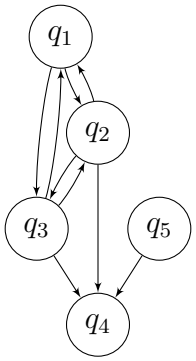



Figure 6.1: A coupling graph for a quantum computer device with 5 qubits.

### 6.3.4 Time

Time, as it is represented within this model, is a unitless ordinal measure. Each time step in this model assumes all gates that are compatible with the device $\mathcal{M}$ can be executed within one time step. In other words, all gates executed on a given hardware take the same amount of time to execute.

This assumption that all gates execute in the same amount of time is not fully representative of the real world. In fact, different gates on different devices take different amounts of time to execute. Future work on how this model may change to accommodate non-uniform gate execution times is explored in section 8.5.

We define the domain for all time in this model by the following range $T$.

$$T = [0..t_{\max}] \tag{6.12}$$

All timing related variables within the constraint model must remain within this bounded region.

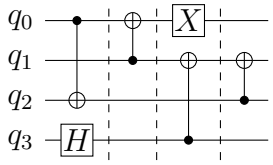



Figure 6.2: A quantum circuit showing the time steps, or layers of the circuit. The vertical dotted lines indicate the separation of of each time step.

## 6.4 Variables

The following is the formal description of the variables used in the constraints within this model. Many of these variables have the same role within the constraints but represent distinct parts of the input constructs. Therefore, an array-like notation is used to denote and differentiate unique variables. The left hand side is an abbreviation of the variable name (such as $GST$ for gate start time) and the right hand side is a list of fields which identify the unique instance of the variable (such as $[\mathcal{G}_i, q_{\text{phys}}]$ for gate start time). Lastly, we will use object notation to access members of the various inputs we provide. For example, the quantum device, $\mathcal{M}$, contains a coupling graph, $C$. We show that we are accessing the device's coupling graph by a period between the two, $\mathcal{M}.C$.

### 6.4.1 Gate Start Time

Each gate must start at a specific time. We introduce an integer variable to represent the time that the gate starts at for each physical qubit:

$$GST[\mathcal{G}_i, q_{\text{phys}}] \in T \quad (\text{Gate Start Time}) \tag{6.13}$$

### 6.4.2 Gate Duration

Gates have an amount of time they actually take to run. This variable denotes the amount of time spent on a physical qubit, $q_{\text{phys}}$, by a particular gate, $\mathcal{G}_i$. We introduce an integer variable to represent the duration of a gate for each physical qubit:

$$GD[\mathcal{G}_i, q_{\text{phys}}] \in T[\mathcal{G}_i, q_{\text{phys}}] \quad (\text{Gate Duration}) \tag{6.14}$$

### 6.4.3 Synthesized Swap Gates

Swap gates may need to be inserted to make a virtual circuit physically realizable. There are several reasons a swap gate may need to be inserted before a gate. Common reasons to

add swap gates to a circuit can be to bring qubits adjacent for an operation and to optimize later operations. Synthesized swap gates are organized onto layers prior to each gate and can only be inserted between qubits with an edge in the device coupling map, $C$. Swap layers identify the order new swap gates are to be inserted in to the physical circuit. There are a fixed number of swap layers for each gate, in this case the number of swap layers for each gate is equal to the number of couplings in the coupling map.

The boolean variable $SGI$ denotes whether a swap gate is to be inserted in the physical circuit for a specific coupling and layer. The integer variable $SGD$ is the sum of swap gate durations of all swaps inserted before a given gate for a certain physical qubit.

$$SGI[\mathcal{G}_i, l, c] \in \{\textsc{True}, \textsc{False}\} \quad (\textsc{Swap Gate Insertion}) \tag{6.15}$$
$$SGD[\mathcal{G}_i, q_{\text{phys}}] \in T \quad (\textsc{Swap Gate Duration}) \tag{6.16}$$
$$\mathcal{G}_i \in \mathcal{C}.\Gamma \tag{6.17}$$
$$l \in [1..|\mathcal{M}.C|] \tag{6.18}$$
$$c \in \mathcal{M}.C \tag{6.19}$$

Note that the number of layers, $l$, is equal to the number of couplings in the device, $\mathcal{M}.C$.

### 6.4.4 Physical-to-Virtual Qubit Mappings

Each circuit can have qubit registers remapped for the purpose of fixing the mapping of physical-to-physical qubits in an efficient manner. The following are the different qubit mappings used:

$$IQM[q_{\text{phys}}] \in \mathcal{R}_{\text{virt}} \cup \{q_\varnothing\} \quad (\textsc{Input Qubit Mapping}) \tag{6.20}$$
$$GQM[\mathcal{G}_i, q_{\text{phys}}] \in \mathcal{R}_{\text{virt}} \cup \{q_\varnothing\} \quad (\textsc{Gate Qubit Mapping}) \tag{6.21}$$
$$SQM[\mathcal{G}_i, l, q_{\text{phys}}] \in \mathcal{R}_{\text{virt}} \cup \{q_\varnothing\} \quad (\textsc{Swap Qubit Mapping}) \tag{6.22}$$

Each of these qubit remappings is used respectively for the circuit input, each gate's input, and each swap layer. Note that each qubit in $\mathcal{R}_{\text{virt}}$ is uniquely mapped to one physical qubit for each of these mappings. Unmapped physical qubits map to a dummy virtual qubit, $q_\varnothing$.

## 6.5 Constraints

The following are the constraints for this model. These constraints, given a circuit $\mathcal{C}$ and device $\mathcal{M}$, model the necessary modifications to the circuit to map it to the device.

### 6.5.1 Bounded Gate Start Time and Duration

Each gate in the input circuit, $\mathcal{C}$, must start at a certain time. More strictly, each gate in the circuit must assign a bounded start time for each of the device's physical qubits.

$$GST[\mathcal{G}_i, q_{\text{phys}}] \in T \tag{6.23}$$
$$0 \leq GST[\mathcal{G}_i, q_{\text{phys}}] \leq \mathcal{M}.t_{\max} \tag{6.24}$$

The same thing is asserted against the gate duration:

$$GD[\mathcal{G}_i, q_{\text{phys}}] \in T \tag{6.25}$$
$$0 \leq GD[\mathcal{G}_i, q_{\text{phys}}] \leq \mathcal{M}.t_{\max} \tag{6.26}$$

For gate duration it is important to note that the duration of a qubit can be zero when the gate does not operate on that qubit.

### 6.5.2 Swap Gate Duration

Sets of synthesized swap gates will run for a certain amount of time. A visual of this is shown in Figure 6.3. The total duration of all of the swap gates prior to a gate, over each physical qubit is given as:

$$SGD[\mathcal{G}_i, q_{\text{phys}}] = \sum_{c \in C} \begin{cases} 1 & SGI[\mathcal{G}_i, l, c] = \textsc{True} \wedge q_{\text{phys}} \in c \\ 0 & \text{otherwise} \end{cases} \tag{6.27}$$

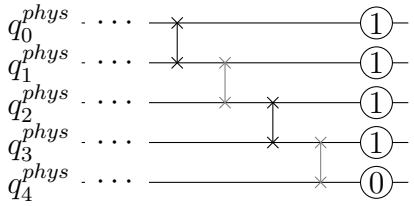



Figure 6.3: A set of synthesized SWAP gates which each consume some amount of time. The active SWAP gates in black consume 1 time step across the top 4 physical qubits.

### 6.5.3 Gate Duration

The gate duration must be at least as long as the gate run time. This run time of a gate is one unit of time. The swap gate duration is added to this value to obtain the total gate duration. This constraint for a single qubit gate is defined as:

$$(GQM[\mathcal{G}_i, q_{\text{phys}}] = q_{\text{gate}}) \rightarrow (GD[\mathcal{G}_i, q_{\text{phys}}] = 1 + SGD[\mathcal{G}_i, q_{\text{phys}}]) \tag{6.28}$$
$$\forall \mathcal{G}_i \in \Gamma \tag{6.29}$$
$$\forall q_{\text{virt}} \in \{q_{\text{virt}} \mapsto GQM[\mathcal{G}_i, q_{\text{phys}}] : \forall q_{\text{gate}} \in \mathcal{G}_i.\mathcal{R}_{\text{gate}}, q_{\text{phys}} \in R_{\text{phys}}\} \tag{6.30}$$

Two qubit gates require more to be constrained than in the single qubit gate case. We need to ensure that all qubits not only take on the latest time of their inputs, but also that all gate outputs must end at the same time. The constraint in this case is as follows:

$$\Big(\bigwedge_{q_\mathrm{j}}\bigwedge_{q_\mathrm{k}} GQM[\mathcal{G}_i, q_\mathrm{j}] = \mathcal{R}_\mathrm{gate}[1] \wedge GQM[\mathcal{G}_i, q_\mathrm{j}] = \mathcal{R}_\mathrm{gate}[2]\Big) \rightarrow \tag{6.31}$$

$$\big(GD[\mathcal{G}_i, q_\mathrm{j}] \geq 1 + SGD[\mathcal{G}_i, q_\mathrm{j}] \tag{6.32}$$

$$\wedge\, GD[\mathcal{G}_i, q_\mathrm{k}] \geq 1 + SGD[\mathcal{G}_i, q_\mathrm{k}] \tag{6.33}$$

$$\wedge\, GD[\mathcal{G}_i, q_\mathrm{j}] \geq 1 + SGD[\mathcal{G}_i, q_\mathrm{k}] \tag{6.34}$$

$$\wedge\, GD[\mathcal{G}_i, q_\mathrm{k}] \geq 1 + SGD[\mathcal{G}_i, q_\mathrm{j}] \tag{6.35}$$

$$\wedge\, GST[\mathcal{G}_i, q_\mathrm{j}] + GD[\mathcal{G}_i, q_\mathrm{j}] = GST[\mathcal{G}_i, q_\mathrm{k}] + GD[\mathcal{G}_i, q_\mathrm{k}]\big) \tag{6.36}$$

$$\forall \mathcal{G}_i \in \Gamma \tag{6.37}$$

$$\forall q_\mathrm{virt} \in \{q_\mathrm{virt} \mapsto GQM[\mathcal{G}_i, q_\mathrm{phys}] : \forall q_\mathrm{gate} \in \mathcal{G}_i, q_\mathrm{phys} \in \mathcal{R}_\mathrm{phys}\} \tag{6.38}$$

$$q_\mathrm{j}, q_\mathrm{k} \in \mathcal{M}.\mathcal{R}_\mathrm{phys} \wedge q_\mathrm{j} \neq q_\mathrm{k} \tag{6.39}$$

### 6.5.4 Gate Input Adjacency

In order to have two qubits *physically* interact as they would in two qubit gates we need them to be *physically close*. We represent this by the device's coupling map. If a directed edge between two qubits exists, then a two qubit gate can be placed there. There are a number of caveats to this statement when we consider certain hardware implementations (such as CZ but not CNOT gates on certain edges), but for now we keep this rule simple and defer expanding it to future work. For single qubit gates, no input adjacency is physically needed, as they only depend on a single qubit. Two qubit gates require this constraint on their inputs to ensure virtual gate qubit inputs are mapped to physically adjacent qubits. We formalize this as follows:

$$\bigvee_{c \in \mathcal{M}.C} (GQM[\mathcal{G}_i, c[1]] = \mathcal{G}_i.\mathcal{R}_\mathrm{gate}[1] \wedge GQM[\mathcal{G}_i, c[2]] = \mathcal{G}_i.\mathcal{R}_\mathrm{gate}[2]) \tag{6.40}$$

$$\forall (\mathcal{G}_i \in \mathcal{C}.\Gamma) \tag{6.41}$$

### 6.5.5 Gate Input Matches Last Swap Layer

A gate's input qubit mapping should inherit the mapping of the last swap layer for that gate. This is to ensure the permutation of qubits in the gate's swap layers are propagated to that gate.

$$GQM[\mathcal{G}_i, q_\mathrm{phys}] = SQM[\mathcal{G}_i, |C|, q_\mathrm{phys}] \tag{6.42}$$

$$\forall (\mathcal{G}_i \in \mathcal{C}.\Gamma) \tag{6.43}$$

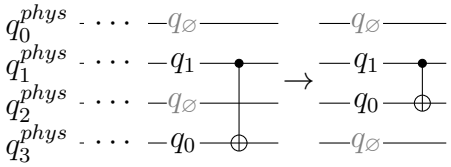



Figure 6.4: On the left is a mapping of a CNOT gate that may not be allowed if there is no coupling between physical qubits $q_1$ and $q_3$. The right shows a possible valid mapping for this gate when there is a coupling between the qubits $q_1$ and $q_2$

.

### 6.5.6 Gate Starts After All Prior Gates Finish

A gate cannot start before its input qubits are done with prior computations. Checking the gate end time (sum of start time and duration) from the prior gate is sufficient because of the topological ordering of the gates.

$$GST[\mathcal{G}_i, q_{\text{phys}}] \geq GST[\mathcal{G}_{i-1}, q_{\text{j}}] + GD[\mathcal{G}_{i-1}, q_{\text{j}}] \tag{6.44}$$

$$\forall(\mathcal{G}_i \in \mathcal{C}.\Gamma \wedge i > 1) \tag{6.45}$$

$$\forall q_{\text{j}} \in \{q_{\text{phys}} \in \mathcal{R}_{\text{phys}} : \forall q_{\text{gate}} \in \mathcal{G}_i.\mathcal{R}_{\text{gate}} \wedge q_{\text{gate}} \mapsto GQM[\mathcal{G}_i, q_{\text{phys}}]\} \tag{6.46}$$

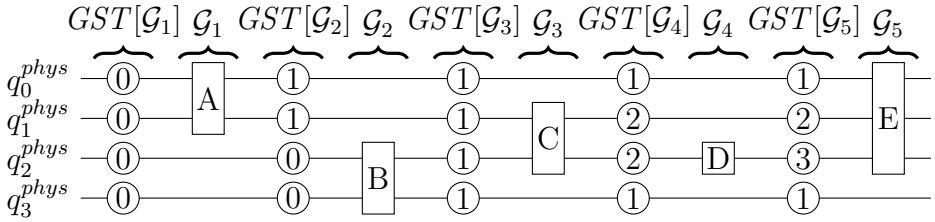



Figure 6.5: A visual example of this constraint. Circles in the circuit are the minimal values for the gate start time.

### 6.5.7 Swap Gate Insertion Unique For Swap Layer

One synthesized swap gate can be accommodated in each swap layer. Either one or no swap gates is allowed to be inserted in a layer. An example of properly assigned sequence of swap gates is shown in Figure 6.6. [1]

$$(\forall \mathcal{G}_i, l)[(\exists! c \in C)[SGI[\mathcal{G}_i, l, c] = \textsc{True}]] \vee [(\nexists c \in C)[SGI[\mathcal{G}_i, l, c] = \textsc{True}]] \tag{6.47}$$

[1]The quantifier notation $\exists!$ denotes not only the requirement for existence but also for uniqueness.

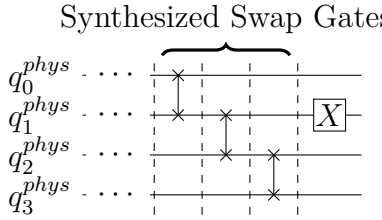



Figure 6.6: The swap layers prior to a gate each with one synthesized swap gate. The dotted lines indicate separate layers.

### 6.5.8 Swap Gate Insertion Swaps Mappings

A swap qubit mapping is going to permute the prior qubit mapping, this will either be a prior swap layer mapping or the circuit input mapping.

$$PQM[\mathcal{G}_i, l, q_{\text{phys}}] = \begin{cases} IQM[q_{\text{phys}}] & l = 1 \wedge i = 1 \\ GQM[\mathcal{G}_{(i-1)}, q_{\text{phys}}] & l = 1 \wedge i > 1 \\ SQM[\mathcal{G}_i, l-1, q_{\text{phys}}] & l > 1 \end{cases} \tag{6.48}$$

$$SQM[\mathcal{G}_i, l, q_{\text{phys}}] = \begin{cases} \textsc{SwapQubits}(PQM[\mathcal{G}_i, l, q_{\text{phys}}], c) & SGI[\mathcal{G}_i, l, q_{\text{phys}} \in c] = \textsc{True} \\ PQM[\mathcal{G}_i, l, q_{\text{phys}}] & SGI[\mathcal{G}_i, l, q_{\text{phys}} \in c] = \textsc{False} \end{cases} \tag{6.49}$$

## 6.6 Optimization Criteria

The optimization criteria within the model can be thought of as special variables. Each of these variables is an objective that the optimization controller sets in each step. The criteria depend on other assignments to variables. This then makes the goal to find an assignment to the other variables which satisfies these criteria.

### 6.6.1 Minimum Number of Swap Gates Inserted

We find the *valid* assignment to the model which minimizes the number of swaps we insert:

$$MSGI(\mathcal{C}_{virt}, \mathcal{M}) = \min \left( \sum_{\mathcal{G}_i \in \Gamma} \sum_{l \in [1..|C|]} \sum_{c \in C} \begin{cases} 1 & SGI[\mathcal{G}_i, l, c] = \textsc{True} \\ 0 & SGI[\mathcal{G}_i, l, c] = \textsc{False} \end{cases} \right) \tag{6.50}$$

### 6.6.2 Minimum Circuit Depth

We find the *valid* assignment to the model which minimizes the overall depth of the circuit. This is also, as of now, the same as saying we find the minimum total circuit end time.

$$MCD(\mathcal{C}_{virt}, \mathcal{M}) = \min\big(\max_{q_{\text{phys}} \in \mathcal{M}.\mathcal{R}_{\text{phys}}} (GST[\mathcal{G}_{|\Gamma|}, q_{\text{phys}}] + GD[\mathcal{G}_{|\Gamma|}, q_{\text{phys}}])\big) \tag{6.51}$$

Currently, we take these criteria and find the *Pareto efficient* solution. This means that we jointly attempt to minimize both criteria until improvements to one will begin to detriment the other. This yields efficient physical circuits.

# Chapter 7

# Results

## 7.1 Introduction

In this chapter we will discuss the results we obtained from the implemented system. This includes both an experimental analysis of the quantum solver as well as the model's ability to optimize quantum programs. An analysis of edge cases which are difficult or impossible for our model to optimize is also provided in a subsequent section. First, we begin with the experimental results.

## 7.2 Experiments

In order to assess this system, there are two metrics used to quantify how well the system performs as a whole. Performance of the system was measured as one of these metrics to understand how a purely classical solver compares to our hybrid solver. A comparative analysis of our optimization strategy against other optimization strategies was formulated to understand how resources are saved in these models. The following discusses both the experimental design and results.

### 7.2.1 Experimental Design

**Performance Analysis** We assume that we are given a random distribution of quantum programs and hardware descriptions as input to our system. As the number of qubits available to our hybrid solver increases, we should see an increase in performance on average as compared to just using the classical solver.

Performance of the hybrid solver is measured by counting the number of classical solver decisions along side the number of Grover's algorithm iterations which are performed. A decision in a backtracking solver such as the one we use, is the event when any variable changes truth value. As we increase the number of qubits we expect the number of Grover's algorithm iterations to increase and the number of decisions decrease. As Grover's algorithm's worst case time complexity for SAT is $\mathcal{O}(2^{n/2})$ and backtracking is $\mathcal{O}(2^n)$, we expect that the number of decisions which are replaced by the quantum solver will indicate a performance improvement.

**Comparative Analysis** The other metric that was quantified was the performance of our optimization model against another system for optimizing quantum programs. For our case, the system chosen was QisKit's Terra framework [32]. Terra is a low-level quantum programming framework with a transpiler collection which, among other things, can perform various levels of program optimization. We will utilize the moderate, normal and aggressive optimizations provided by the transpiler and assess the difference from our model by the reduction of the runtime of the circuit from the original circuit and the change in number of gates.

### 7.2.2 Experimental Results

The following are the results of the experiments laid out above. Both an analysis of the system's performance and comparison to QisKit is provided. All experiments were run on a machine running `Ubuntu 18.04` on an `AMD EPYC 7502P 32-Core` processor with 256 gigabytes of system memory.

#### Performance Analysis

The performance of the hybrid system was analyzed by varying our simulated quantum solver's number of available qubits and then measuring the performance of the classical and quantum solver. We measured these performance metrics by counting the number of decisions made by the classical SAT solver along with the number of Grover iterations required by the quantum solver. In order to analyze this performance we randomly generated a sample of 10 circuits and coupling graph pairs. In this sample we fixed the number of gates and the number of qubits to 5 and 3 respectively. The coupling graphs were each a random linear arrangement. The number of available qubits was varied from 0 to 30. In Figure 7.1 we can see the results of this experiment.

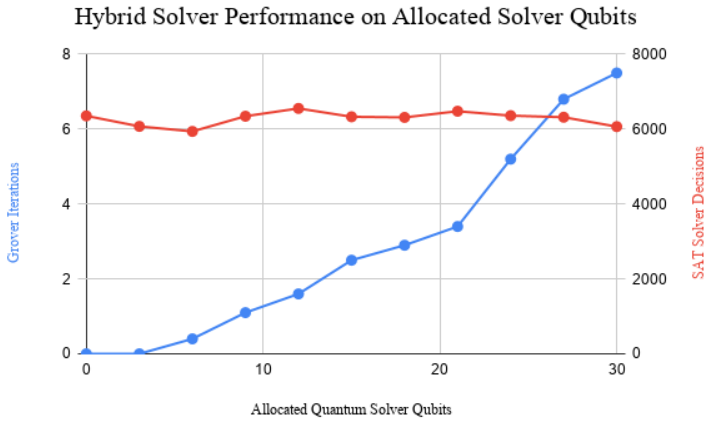



Figure 7.1: The average number of Grover iterations and SAT decisions when varying the number of solver qubits.

As can be seen in Figure 7.1, even for a small circuits and hardware configurations the number of SAT decisions exceeds the number of possible Grover iterations by several orders

of magnitude. That being considered, we also measured the number of SAT decisions which were replaced by a number of Grover iterations. This is displayed in Figure 7.2.

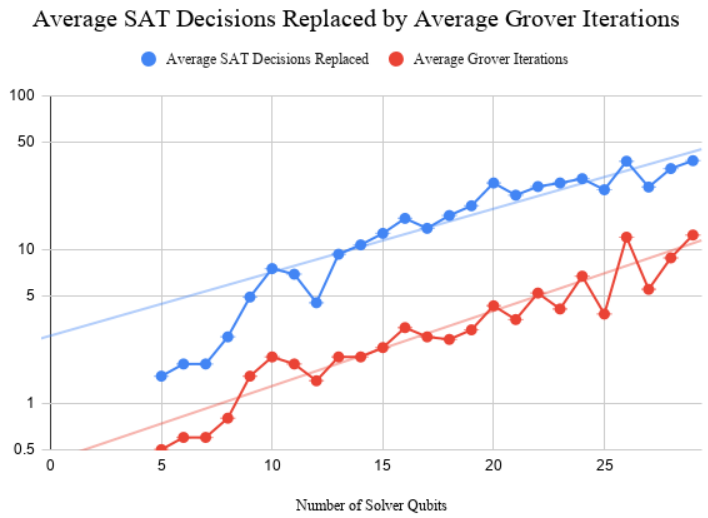



Figure 7.2: A logarithmic plot of the average number of Grover iterations replacing some number of SAT decisions when varying the number of solver qubits.

In the above figure it can be seen that there is a polynomial speedup from the classical solver to the quantum solver on these smaller subproblems. It is clear from these graphs that although the number of solver qubits increasing denotes an increase in steps taken by the quantum solver, this does not have any measurable affect on performance for circuits of this size. This is expected as the ratio of Boolean variables to qubits is too low to see such changes as is predicted in our discussion of the solver limitations in section 5.2.

### Comparative Analysis

The comparative analysis of our optimization model against QisKit's transpiler optimizations measured two qualities of the output circuit: depth and total gates. Depth is a measure of the runtime of the program and the total gate count is another important cost measure. Both are costs or criteria our model optimizes for. Using a similar sample of circuits and coupling graphs as in our performance analysis, we run our model against the 4 optimization levels provided QisKit, obtaining the output circuit of each run. QisKit's level 0 optimization is supposed to simply map the circuit to the coupling graph with minimal effort while level 3 is supposed to offer the highest level of optimization.

In Figure 7.3 we show the average percentage increase in depth after each optimization scheme has been applied to the input circuit. Similarly, Figure 7.4 shows the average percentage increase in gate count after each optimization scheme has been applied to the input circuit.

We can see that our model succeeds in this task as it is able to provide a highly optimized circuit whereas the QisKit optimizations do not achieve the same success. This is likely due

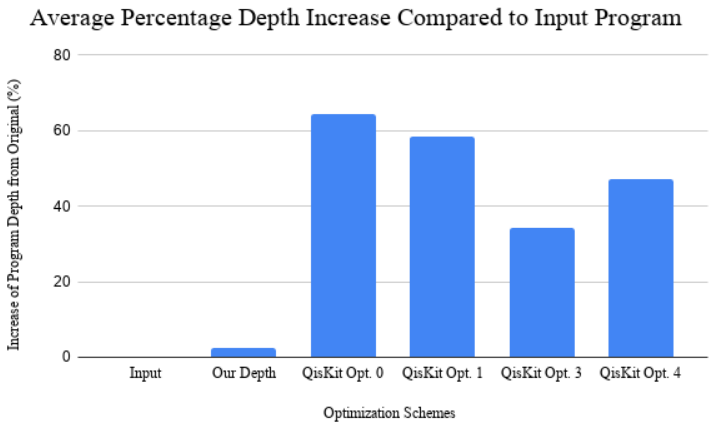



Figure 7.3: The average percentage increase in depth between schemes.

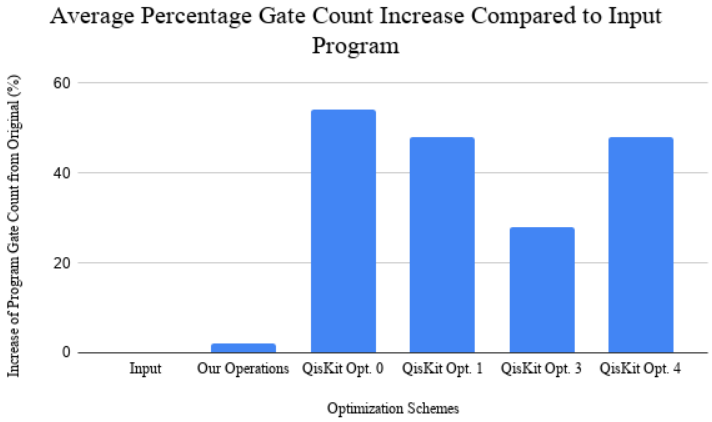



Figure 7.4: The average percentage increase in gate count between schemes.

to the fact that QisKit does not optimize the reordering of the input qubit register whereas our model allows such remapping. Also, QisKit uses an approximate scheme for reducing circuit depth when routing gates, leading to extraneous SWAP gates being inserted. Overall, by these metrics our method is successful but improvements to our procedure are necessary to allow larger circuits, which QisKit can cope with, to be feasibly optimized.

## 7.3 Interesting Instances

Several instances or edge cases were encountered which illustrate the limitations of this system. These edges cases are inputs to our system which fall into two categories: hard edge cases and impossible instances. Hard cases are those which can be optimized but the solver is going to spend a significant amount of time on this. Impossible instances are those which

our model does not support finding the optimal solutions which are trivial to optimize. The next subsections outline a few salient examples of these two cases.

### 7.3.1 Hard Instances

As previously stated, these hard instances are those which are solvable within the model but may be difficult for the solver to arrive at a solution.

One of the hard cases we identified was the Toffoli gate. These three-qubit gates cannot be directly implemented on quantum computers and must be decomposed into many gates. This decomposition is shown in Figure 7.5. This is not particularly hard when the number of physical qubits is three but becomes challenging when more physical qubits and couplings are added as in Figure 7.6. It follows that multiple of these gates in series would also be very challenging to optimize. A potential solution to this would be to introduce the notion of a subroutine into our model to allow those routines to be optimized as a subproblem and then optimized within the larger circuit.

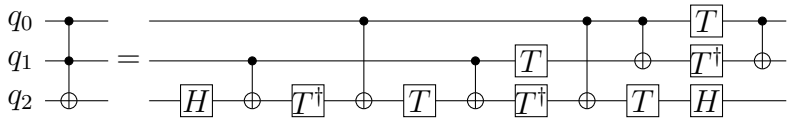



Figure 7.5: A Toffoli, sometimes called a CCNOT or CCX gate, and one of its common decompositions.

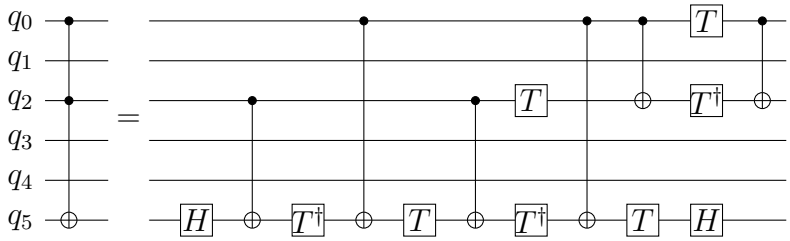


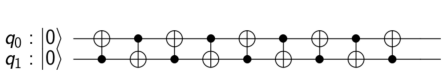

Figure 7.6: A Toffoli gate spanning multiple qubits.

### 7.3.2 Impossible Instances

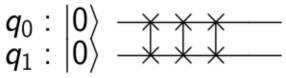
The impossible to optimize cases display the limitations of the model in its current state. Future directions for this work should aim to address the more general issue displayed by these examples.

One case that was identified was that our model does not support analysis of gate commutation. If gates are allowed to commute then we would be able to swap pairs of gates to better utilize each qubit at each time step.

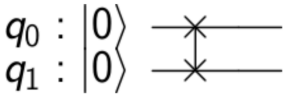
Another case we identified was gate composition and decomposition. Consider a quantum computing hardware which directly supports the SWAP gate. Also consider a user provides a program to be run on the hardware where they have decomposed the SWAP gates in their



(a) An example circuit a programmer may provide containing nine alternating CNOT gates.



(b) A composition of the nine CNOT gates in Figure 7.7a into three consecutive SWAP gates.



(c) A further composition of Figure 7.7b into one SWAP gate.

Figure 7.7: An example of composition as described in subsection 7.3.2.

circuit into CNOT gates, such as in Figure 7.7a. This is action on behalf of the user was unnecessary given that the hardware supports the SWAP operation. An intermediate system such as ours should cope with these user mistakes by composing these gates as seen step wise in Figure 7.7b down to a single SWAP gate in Figure 7.7c.

Gate decomposition is the inverse of this issue, and our model also does not cope with it. Gate decomposition would explore sets of gates which could implement an unsupported operation on a given hardware. For example, suppose we have a controlled Z gate (CZ) with the control on logical qubit 0 and the target on logical qubit 1. Suppose the program optimizer knows that CZ gates are not supported on the hardware directly and that controlled gates can only be placed with the control on physical qubit 1 and the target on physical qubit 0. The optimizer has several options:

- It could simply remap the logical qubits so that the CZ gate is mapped to the hardware correctly, then decompose the CZ gate to gates which are supported by the hardware.
- It could decompose the CZ gate into several gates that could be more resource efficient for the circuit as a whole.

Neither of these options would be fully supported by our system as it exists now as we do not have a way to model these gate decompositions yet. Modeling this task would involve allowing new gates to be inserted on the fly, changing fundamentally how gates, time, and qubit mappings are formulated within our model.

# Chapter 8

# Future Work

## 8.1 Introduction

Initially the goal of this system was to build a self-hosting quantum program optimization system as a proof-of-concept. Although this has been done, there are many areas to improve the proof-of-concept design to a more capable system. The following sections touch on ideas for the future of this work.

## 8.2 Improvements to Hybrid Quantum-Classical Solver

The argument that we posit is that we can speed up the optimization procedure through using quantum algorithms to host the optimization procedure. Yet, the analysis in section 5.2 has shown that we cannot practically speedup this architecture without many more qubits using Grover's algorithm. The aim of implementing this hybrid solver realizes self-hosting but improvements to realize speedups of this process remains to be made. Future work could implement a new replacement to the current quantum solver based on any a variety of quantum backtracking algorithms [9, 10, 22]. The current theory supports a potential speedup over classical equivalent algorithms for a quantum backtracking solver.

## 8.3 Modeling Gate Commutation

Gate commutation is a very promising optimization strategy for reducing quantum circuit depth. This quantum circuit optimization technique is introduced and employed by Itoko et al. [7] but only commutes gates in an effort to find a local optimum circuit under their model. We would aim to do this in an effort to globally improve circuit performance. This property of the gates in a quantum circuit allow us to commute two gates if doing so results in a reduced gate depth. Two gates commute if their unitary operations, rather the unitary matrices, commute.

$$U_1 U_2 = U_2 U_1 \tag{8.1}$$

$$U_1, U_2 \in \mathbb{C}^{2^n} \times \mathbb{C}^{2^n} \tag{8.2}$$

For example, consider a circuit with a Hadamard gate followed by a CNOT gate as is shown in the colored gates in Figure 8.1. Such a circuit could have these two gates commuted within a larger program in order to decrease total circuit depth. An example of this is given visually in Figure 8.1.

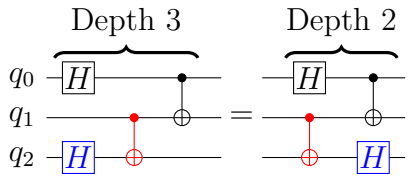



Figure 8.1: An example of a quantum program which can be reduced in depth by applying commutation of the blue and red gates.

Mathematically, an inspection of the unitary matrices of these gates reveals whether it is possible for this gate pair to commute.

$$(I \otimes H) * \text{CNOT} = \text{CNOT} * (I \otimes H) \tag{8.3}$$

If the equality holds then we can consider commuting these gates in the constraint model. This would allow the gates to be swapped, potentially increasing the number of gates which can run at the same time. This would decrease the depth of the circuit overall. For any given circuit, this commutativity property of two gates would need to be determined for each unique pair of unitary operations that appear in the circuit. The results of this check could then be considered in the constraint model. Considerations for how this is to be formalized will be defined in the future.

## 8.4 Modeling Physical Hardware Noise

Quantum computing hardware will always be susceptible to noise from the environment. This quantum circuit optimization model should be able to capture and account for this. Depending on the device used, different quantum gates have different probability of success. We expect the quantum computer device manufacturer to provide high quality experimental analysis of this error rate as part of the device specification. This data can then be incorporated into the constraint model to inform the optimization procedure. In turn this would produce circuits that yield more accurate measurement results given we consider overall measurement error as a new optimization criteria.

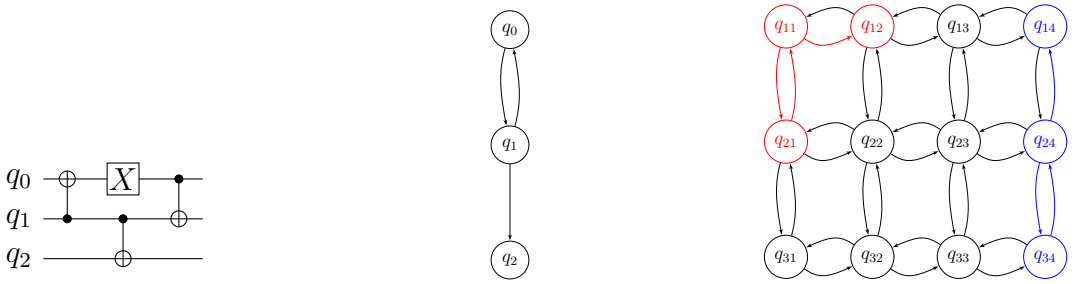
## 8.5 Modeling Gate Timing

Similar to how various gates have different noise and error rates, different gates available on the same hardware take different amounts of time to complete. For example, the relative time needed to compute CNOT gates is typically longer than single qubit gates, such as Hadamard and Pauli-X gates. Modeling this by taking a configuration of the timings for each gate supported on a hardware will allow us to better represent optimized circuits.

Further, such timings would be useful in generating a more accurate schedule for the gates. This is something our model does not accommodate for, as we assume all gates take one unit of time each.

## 8.6 Simplifying Coupling Constraints through Inspection of Sub-graphs

Let us consider the hardware coupling map for a feasible quantum computer. In this coupling map is a graph represented by a mesh of qubits such as can be seen in Figure 8.2. If our circuit utilizes a subset of available qubits, we can map this circuit to many isomorphic subgraphs of the coupling map. This reduces the number of such subgraphs inspected by the solver. This has the potential to reduce the problem size, pruning the assignment search space.




(a) A simple circuit with several two qubit gates.

(b) A graph of the connections between qubits in the circuit.

(c) A coupling graph with two subgraphs in red and blue that the circuit could be mapped to.

Figure 8.2: An example of the subgraph isomorphism issue described in this section.

Note that the subgraph isomorphism problem is reducible to a satisfiability problem, which we can run using the quantum satisfiability solvers discussed prior. We can run this for specific problem instances and cache the results for even faster later use in similar instances.

## 8.7 Efficient Gate Compositions and Decompositions and Similar Cost Reductions

Several approaches to quantum program optimization examine the composition and decomposition of various gates [5–7, 33]. A set of quantum gate can be composed if the product of those gates produces a new single gate that is compatible with the target quantum device. Gate decomposition works similarly, but one gate produces several compatible gates. This can be shown formally via the products of the unitary operations:

$$U = \prod_{i=1}^{m} U_i \tag{8.4}$$

$$U, U_i \in \mathbb{C}^{2^n} \times \mathbb{C}^{2^n} \tag{8.5}$$

We hope to employ some of these notions into our constraint model for preprocessing circuits prior to our optimization procedure. One possible decomposition of a SWAP gate is shown in Figure 8.3. In Figure 8.4 an example of a cost efficient decomposition of a bridge gate is shown as presented by Itoko et al. [7].

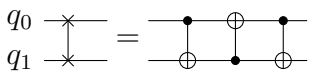



Figure 8.3: The decomposition of a SWAP gate into 3 consecutive CNOT gates.

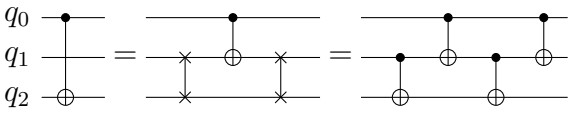



Figure 8.4: A CNOT with span across another qubit and a series of decompositions of the gate. From left to right: 1) A CNOT gate spanning over an intermediary qubit. 2) the swap based approach, which may not be supported on all hardware implementations and is costly, possibly decomposing to a circuit of depth 7, and 3) The bridge gate approach to this CNOT. Note that it only requires CNOT gates in one direction and is depth 4.

## 8.8 Modeling Subroutines

In many cases certain patterns of gates are used multiple times in a given quantum program. In our model, we could simplify the program by modeling subroutines in the quantum solver. By first optimizing these repeated patterns, or subroutines, and then using the produced subroutine circuits we can insert in place after optimizing the program surround the subroutines. This would reduce the runtime of the optimization process for many practical circuits which have many repeated components.

## 8.9 Configurable Hardware Model

We would like to improve the current system's usability by providing a convenient mechanism to describe the desired quantum hardware. This would allow the user to give a specification of the hardware just as the user can already easily specify the input quantum program.

# Chapter 9

# Conclusion

A system has been presented for self-hosted quantum program optimization. We use a constraint model and quantum satisfiability algorithms to find optimized quantum programs. Other work has not attempted to show a self-hosted quantum optimization process, but this work begins the discussion of such topics. We have made the source code for this project available for anyone to use.

The source code for this work can be found at `https://github.com/CantelopePeel/QQ`. Within this repository you can find the necessary scripts to set up this framework as well as a step by step example of the process we implement. Auxiliary repositories for our modified version of the Z3 SMT solver and QuEST quantum simulator can be found at `https://github.com/CantelopePeel/Z3` and `https://github.com/CantelopePeel/QuEST`, respectively.

This model and the architecture as a whole are a leading step towards efficient cost reduction measures for quantum computation, using quantum computation itself to achieve that end.

# Chapter 10

# Appendix A: System Architecture

Below is a diagram depicting our full system architecture as describe in the primary matter of this thesis.

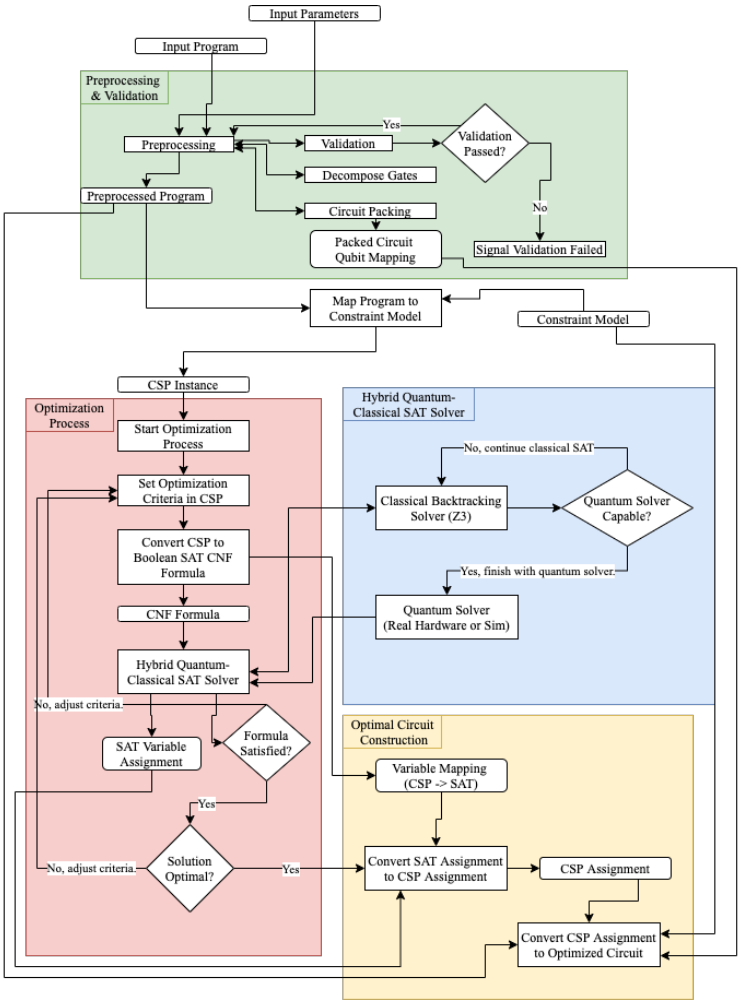



Figure 10.1: System architecture diagram.

# Bibliography


[1] L. K. Grover, "A fast quantum mechanical algorithm for database search," *arXiv:quant-ph/9605043*, Nov. 1996. arXiv: quant-ph/9605043.

[2] P. Murali, A. Javadi-Abhari, F. T. Chong, and M. Martonosi, "Formal Constraint-based Compilation for Noisy Intermediate-Scale Quantum Systems," *Microprocessors and Microsystems*, vol. 66, pp. 102–112, Apr. 2019. arXiv: 1903.03276.

[3] M. Pedram and A. Shafaei, "Layout Optimization for Quantum Circuits with Linear Nearest Neighbor Architectures," *IEEE Circuits and Systems Magazine*, vol. 16, no. 2, pp. 62–74, 2016.

[4] G. Meuli, M. Soeken, and G. De Micheli, "SAT-based {CNOT, T} Quantum Circuit Synthesis," in *Reversible Computation* (J. Kari and I. Ulidowski, eds.), Lecture Notes in Computer Science, (Cham), pp. 175–188, Springer International Publishing, 2018.

[5] D. Maslov, G. W. Dueck, D. M. Miller, and C. Negrevergne, "Quantum Circuit Simplification and Level Compaction," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, pp. 436–444, Mar. 2008. arXiv: quant-ph/0604001.

[6] X. Zhang, H. Xiang, T. Xiang, L. Fu, and J. Sang, "An efficient quantum circuits optimizing scheme compared with QISKit," *arXiv:1807.01703 [quant-ph]*, July 2018. arXiv: 1807.01703.

[7] T. Itoko, R. Raymond, T. Imamichi, and A. Matsuo, "Optimization of Quantum Circuit Mapping using Gate Transformation and Commutation," *arXiv:1907.02686 [quant-ph]*, July 2019. arXiv: 1907.02686.

[8] A. Zulehner, A. Paler, and R. Wille, "An Efficient Methodology for Mapping Quantum Circuits to the IBM QX Architectures," *arXiv:1712.04722 [quant-ph]*, June 2018. arXiv: 1712.04722.

[9] A. Montanaro, "Quantum walk speedup of backtracking algorithms," *arXiv:1509.02374 [quant-ph]*, Sept. 2015. arXiv: 1509.02374.

[10] A. Ambainis and M. Kokainis, "Quantum algorithm for tree size estimation, with applications to backtracking and 2-player games," *arXiv:1704.06774 [quant-ph]*, Apr. 2017. arXiv: 1704.06774.

[11] A. W. Cross, L. S. Bishop, J. A. Smolin, and J. M. Gambetta, "Open Quantum Assembly Language," *arXiv:1707.03429 [quant-ph]*, July 2017. arXiv: 1707.03429.

[12] S. A. Cook, "The complexity of theorem-proving procedures," in *Proceedings of the third annual ACM symposium on Theory of computing*, STOC '71, (Shaker Heights, Ohio, USA), pp. 151–158, Association for Computing Machinery, May 1971.

[13] "Z3Prover/z3," Dec. 2019. original-date: 2015-03-26T18:16:07Z.

[14] D. Venturelli, M. Do, E. Rieffel, and J. Frank, "Compiling quantum circuits to realistic hardware architectures using temporal planners," *Quantum Science and Technology*, vol. 3, p. 025004, Feb. 2018.

[15] M. Soeken, G. Meuli, B. Schmitt, F. Mozafari, H. Riener, and G. De Micheli, "Boolean satisfiability in quantum compilation," *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 378, p. 20190161, Feb. 2020.

[16] N. J. Cerf, L. K. Grover, and C. P. Williams, "Nested quantum search and NP-complete problems," *Physical Review A*, vol. 61, p. 032303, Feb. 2000. arXiv: quant-ph/9806078.

[17] L. K. Grover and J. Radhakrishnan, "Is partial quantum search of a database any easier?," *arXiv:quant-ph/0407122*, July 2004. arXiv: quant-ph/0407122.

[18] Vedran Dunjko, Y. Ge, and J. I. Cirac, "Computational speedups using small quantum devices," July 2018.

[19] E. Dantsin, V. Kreinovich, and A. Wolpert, "On Quantum Versions of Record-breaking Algorithms for SAT," *SIGACT News*, vol. 36, pp. 103–108, Dec. 2005.

[20] E. Dantsin and A. Wolpert, *Quantum Versions of k-CSP Algorithms: a First Step Towards Quantum Algorithms for Interval-Related Constraint Satisfaction Problems*.

[21] E. Campbell, A. Khurana, and A. Montanaro, "Applying quantum algorithms to constraint satisfaction problems," *Quantum*, vol. 3, p. 167, July 2019. arXiv: 1810.05582.

[22] S. Martiel and M. Remaud, "Practical implementation of a quantum backtracking algorithm," *arXiv:1908.11291 [quant-ph]*, Aug. 2019. arXiv: 1908.11291.

[23] A. Montanaro, "Quantum speedup of branch-and-bound algorithms," *arXiv:1906.10375 [quant-ph]*, June 2019. arXiv: 1906.10375.

[24] E. Farhi, J. Goldstone, and S. Gutmann, "A Quantum Approximate Optimization Algorithm," *arXiv:1411.4028 [quant-ph]*, Nov. 2014. arXiv: 1411.4028.

[25] B. Apolloni, C. Carvalho, and D. de Falco, "Quantum stochastic optimization," *Stochastic Processes and their Applications*, vol. 33, pp. 233–244, Dec. 1989.

[26] Kuk-Hyun Han and Jong-Hwan Kim, "Quantum-inspired evolutionary algorithm for a class of combinatorial optimization," *IEEE Transactions on Evolutionary Computation*, vol. 6, pp. 580–593, Dec. 2002.

[27] G. S. Tseitin, "On the Complexity of Derivation in Propositional Calculus," in *Automation of Reasoning: 2: Classical Papers on Computational Logic 1967–1970* (J. H. Siekmann and G. Wrightson, eds.), Symbolic Computation, pp. 466–483, Berlin, Heidelberg: Springer, 1983.

[28] "Using Grover's Algorithm | CNOT."

[29] T. Schoning, "A probabilistic algorithm for k-SAT and constraint satisfaction problems," in *40th Annual Symposium on Foundations of Computer Science (Cat. No.99CB37039)*, pp. 410–414, Oct. 1999. ISSN: 0272-5428.

[30] E. Dantsin, A. Goerdt, E. A. Hirsch, R. Kannan, J. Kleinberg, C. Papadimitriou, P. Raghavan, and U. Schöning, "A deterministic (2−2/(k+1))n algorithm for k-SAT based on local search," *Theoretical Computer Science*, vol. 289, pp. 69–83, Oct. 2002.

[31] "QuEST-Kit/QuEST," Mar. 2020. original-date: 2017-03-28T12:00:18Z.

[32] "Qiskit/qiskit-terra," Apr. 2020. original-date: 2017-03-03T17:02:42Z.

[33] M. Mottonen and J. J. Vartiainen, "Decompositions of general quantum gates," *arXiv:quant-ph/0504100*, Apr. 2005. arXiv: quant-ph/0504100.

[34] P. J. Coles, S. Eidenbenz, S. Pakin, A. Adedoyin, J. Ambrosiano, P. Anisimov, W. Casper, G. Chennupati, C. Coffrin, H. Djidjev, D. Gunter, S. Karra, N. Lemons, S. Lin, A. Lokhov, A. Malyzhenkov, D. Mascarenas, S. Mniszewski, B. Nadiga, D. O'Malley, D. Oyen, L. Prasad, R. Roberts, P. Romero, N. Santhi, N. Sinitsyn, P. Swart, M. Vuffray, J. Wendelberger, B. Yoon, R. Zamora, and W. Zhu, "Quantum Algorithm Implementations for Beginners," *arXiv:1804.03719 [quant-ph]*, Apr. 2018. arXiv: 1804.03719.

[35] T. Häner, D. S. Steiger, K. Svore, and M. Troyer, "A software methodology for compiling quantum programs," *Quantum Science and Technology*, vol. 3, p. 020501, Feb. 2018.

[36] A. Leporati and S. Felloni, "Three "Quantum" Algorithms to Solve 3-SAT," *Theor. Comput. Sci.*, vol. 372, pp. 218–241, Mar. 2007.

[37] D. S. Steiger, T. Häner, and M. Troyer, "ProjectQ: An Open Source Software Framework for Quantum Computing," *Quantum*, vol. 2, p. 49, Jan. 2018. arXiv: 1612.08091.

[38] S. Achour and M. Rinard, "Time Dilation and Contraction for Programmable Analog Devices with Jaunt," in *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems - ASPLOS '18*, (Williamsburg, VA, USA), pp. 229–242, ACM Press, 2018.

[39] A. Solar-Lezama, L. Tancau, R. Bodik, S. Seshia, and V. Saraswat, "Combinatorial Sketching for Finite Programs," in *Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XII, (New York, NY, USA), pp. 404–415, ACM, 2006. event-place: San Jose, California, USA.

[40] S. Achour, R. Sarpeshkar, and M. C. Rinard, "Configuration Synthesis for Programmable Analog Devices with Arco," in *Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '16, (New York, NY, USA), pp. 177–193, ACM, 2016. event-place: Santa Barbara, CA, USA.

[41] J.-P. Watson, C. A. Phillips, and R. D. Carr, "Solving a Scheduling Problem for a Quantum Computing Architecture Using Constraint Programming.," Tech. Rep. SAND2008-7438C, Sandia National Lab. (SNL-NM), Albuquerque, NM (United States), Nov. 2008.

[42] Y. Nam, N. J. Ross, Y. Su, A. M. Childs, and D. Maslov, "Automated optimization of large quantum circuits with continuous parameters," *npj Quantum Information*, vol. 4, pp. 1–12, May 2018.

[43] A. Parent, M. Roetteler, and K. M. Svore, "Reversible circuit compilation with space constraints," *arXiv:1510.00377 [quant-ph]*, Oct. 2015. arXiv: 1510.00377.

[44] C. A. Trugenberger, "Quantum optimization for combinatorial searches," *New Journal of Physics*, vol. 4, pp. 26–26, Apr. 2002.

[45] S.-T. Cheng and M.-H. Tao, "Quantum cooperative search algorithm for 3-SAT," *Journal of Computer and System Sciences*, vol. 73, pp. 123–136, Feb. 2007.

[46] C. P. Gomes, H. Kautz, A. Sabharwal, and B. Selman, "Chapter 2 Satisfiability Solvers," in *Foundations of Artificial Intelligence*, vol. 3, pp. 89–134, Elsevier, 2008.

[47] F. G. S. L. Brandao and K. Svore, "Quantum Speed-ups for Semidefinite Programming," Sept. 2016.

[48] P. Murali, J. M. Baker, A. Javadi-Abhari, F. T. Chong, and M. Martonosi, "Noise-Adaptive Compiler Mappings for Noisy Intermediate-Scale Quantum Computers," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '19, (New York, NY, USA), pp. 1015–1029, ACM, 2019. event-place: Providence, RI, USA.

[49] S. M. Saeed, X. Cui, R. Wille, A. Zulehner, K. Wu, R. Drechsler, and R. Karri, "Towards Reverse Engineering Reversible Logic," *arXiv:1704.08397 [cs]*, Apr. 2017. arXiv: 1704.08397.

[50] P. Murali, N. M. Linke, M. Martonosi, A. J. Abhari, N. H. Nguyen, and C. H. Alderete, "Full-stack, real-system quantum computer studies: architectural comparisons and design insights," in *Proceedings of the 46th International Symposium on Computer Architecture - ISCA '19*, (Phoenix, Arizona), pp. 527–540, ACM Press, 2019.

[51] M. Side and V. Erol, "Applying Quantum Optimization Algorithms for Linear Programming," Apr. 2017.

[52] J. Petke and P. Jeavons, "The Order Encoding: From Tractable CSP to Tractable SAT," in *Theory and Applications of Satisfiability Testing - SAT 2011* (K. A. Sakallah and L. Simon, eds.), vol. 6695, pp. 371–372, Berlin, Heidelberg: Springer Berlin Heidelberg, 2011.

[53] A. Ignatiev, J. Marques-Silva, and A. Morgado, "PySAT Documentation," p. 73.

[54] J. Liu, B. Zhan, S. Wang, S. Ying, T. Liu, Y. Li, M. Ying, and N. Zhan, "Formal Verification of Quantum Algorithms Using Quantum Hoare Logic," in *Computer Aided Verification* (I. Dillig and S. Tasiran, eds.), Lecture Notes in Computer Science, pp. 187–207, Springer International Publishing, 2019.

[55] M. Santha, "Quantum walk based search algorithms," *arXiv:0808.0059 [quant-ph]*, Aug. 2008. arXiv: 0808.0059.

[56] Z. Bian, F. Chudak, R. Israel, B. Lackey, W. G. Macready, and A. Roy, "Mapping constrained optimization problems to quantum annealing with application to fault diagnosis," *arXiv:1603.03111 [quant-ph]*, Mar. 2016. arXiv: 1603.03111.

[57] N. de Beaudrap and S. Gharibian, "A linear time algorithm for quantum 2-SAT," *arXiv:1508.07338 [quant-ph]*. arXiv: 1508.07338.

[58] I. Arad, M. Santha, A. Sundaram, and S. Zhang, "Linear-Time Algorithm for Quantum 2SAT," *Theory of Computing*, vol. 14, pp. 1–27, Mar. 2018.

[59] S. Bravyi, "Efficient algorithm for a quantum analogue of 2-SAT," *arXiv:quant-ph/0602108*, Feb. 2006. arXiv: quant-ph/0602108.

[60] E. Farhi, S. Kimmel, and K. Temme, "A Quantum Version of Sch\"oning's Algorithm Applied to Quantum 2-SAT," *arXiv:1603.06985 [quant-ph]*, Mar. 2016. arXiv: 1603.06985.

[61] A. Ambainis, "Quantum walk algorithm for element distinctness," *arXiv:quant-ph/0311001*, Oct. 2003. arXiv: quant-ph/0311001.

[62] T. Hogg, "Adiabatic Quantum Computing for Random Satisfiability Problems," *Physical Review A*, vol. 67, p. 022314, Feb. 2003. arXiv: quant-ph/0206059.

[63] A. Ambainis, "Quantum search algorithms," *arXiv:quant-ph/0504012*, Apr. 2005. arXiv: quant-ph/0504012.

[64] W. L. Yang, H. Wei, F. Zhou, W. L. Chang, and M. Feng, "Solution to Satisfiability problem by a complete Grover search with trapped ions," *Journal of Physics B: Atomic, Molecular and Optical Physics*, vol. 42, p. 145503, July 2009. arXiv: 0811.2905.

[65] J. HV, H. Thapliyal, H. R. Arabnia, and V. K. Agrawal, "Ancilla-Input and Garbage-Output Optimized Design of a Reversible Quantum Integer Multiplier," *arXiv:1608.01228 [quant-ph]*, Aug. 2016. arXiv: 1608.01228.

[66] Y. Wang, "A Quantum Walk Enhanced Grover Search Algorithm for Global Optimization," *arXiv:1711.07825 [quant-ph]*, Nov. 2017. arXiv: 1711.07825.

[67] A. de Araújo and M. Finger, "Classical and quantum satisfiability," *Electronic Proceedings in Theoretical Computer Science*, vol. 81, pp. 79–84, Mar. 2012. arXiv: 1203.6161.

[68] A. Younes and J. E. Rowe, "A Polynomial Time Bounded-error Quantum Algorithm for Boolean Satisfiability," *arXiv:1507.05061 [cs]*, July 2015. arXiv: 1507.05061.

[69] G. F. Viamontes, I. L. Markov, and J. P. Hayes, "Is Quantum Search Practical?," *arXiv:quant-ph/0405001*, Apr. 2004. arXiv: quant-ph/0405001.

[70] D. Bacon, "CSE 599 d-Quantum Computing The Quantum Circuit Model and Universal Quantum Computation," 2006.

[71] A. M. Childs, B. W. Reichardt, R. Spalek, and S. Zhang, "Every NAND formula of size N can be evaluated in time N^{1/2+o(1)} on a quantum computer," *arXiv:quant-ph/0703015*, Mar. 2007. arXiv: quant-ph/0703015.

[72] A. Ambainis, A. M. Childs, B. W. Reichardt, R. Špalek, and S. Zhang, "Any AND-OR Formula of Size $N$ Can Be Evaluated in Time $N^{1/2+o(1)}$ on a Quantum Computer," *SIAM Journal on Computing*, vol. 39, pp. 2513–2530, Jan. 2010.

[73] D. Gottesman and I. L. Chuang, "Quantum Teleportation is a Universal Computational Primitive," *Nature*, vol. 402, pp. 390–393, Nov. 1999. arXiv: quant-ph/9908010.

[74] D. Deutsch and R. Jozsa, "Rapid solution of problems by quantum computation," *Proceedings of the Royal Society of London. Series A: Mathematical and Physical Sciences*, vol. 439, pp. 553–558, Dec. 1992.

[75] Y. Ge and V. Dunjko, "A hybrid algorithm framework for small quantum computers with application to finding Hamiltonian cycles," *Journal of Mathematical Physics*, vol. 61, p. 012201, Jan. 2020.

[76] G. G. Guerreschi and M. Smelyanskiy, "Practical optimization for hybrid quantum-classical algorithms," *arXiv:1701.01450 [quant-ph]*, Jan. 2017. arXiv: 1701.01450.

[77] I. Lynce and J. Marques-Silva, "Efficient data structures for backtrack search SAT solvers," *Annals of Mathematics and Artificial Intelligence*, vol. 43, pp. 137–152, Jan. 2005.

[78] H. Riener, R. Ehlers, B. Schmitt, and G. De Micheli, "Exact Synthesis of ESOP Forms," *arXiv:1807.11103 [cs]*, July 2018. arXiv: 1807.11103.

[79] C. Dumitrescu, "A randomized, efficient algorithm for 3SAT," *arXiv:1703.01905 [cs]*, Jan. 2018. arXiv: 1703.01905.

[80] A. Layeb and D.-E. Saidouni, "A Hybrid Quantum Genetic Algorithm and Local Search based DPLL for Max 3-SAT Problems," *Applied Mathematics & Information Sciences*, vol. 8, pp. 77–87, Jan. 2014.

[81] C. Shao, Y. Li, and H. Li, "Quantum Algorithm Design: Techniques and Applications," *Journal of Systems Science and Complexity*, vol. 32, pp. 375–452, Feb. 2019.

[82] "Qiskit/qiskit," Apr. 2020. original-date: 2018-12-12T22:04:07Z.