3-2021

Solving the Quantum Layout Problem for NISQ-Era Quantum Computers via Metaheuristic Algorithms

Brian D. Curran Jr.

Solving the Quantum Layout Problem for NISQ-Era Quantum Computers via Metaheuristic Algorithms

THESIS

Brian D. Curran Jr., Second Lieutenant, USAF

AFIT-ENG-MS-21-M-024

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT-ENG-MS-21-M-024

SOLVING THE QUANTUM LAYOUT PROBLEM FOR NISQ-ERA QUANTUM COMPUTERS VIA METAHEURISTIC ALGORITHMS

THESIS

Presented to the Faculty

Department of Electrical and Computer Engineering

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

in Partial Fulfillment of the Requirements for the

Degree of Master of Science in Computer Science

Brian D. Curran Jr., BS

Second Lieutenant, USAF

March 2021

DISTRIBUTION STATEMENT A

APPROVED FOR PUBLIC RELEASE, DISTRIBUTION UNLIMITED.

AFIT-ENG-MS-21-M-024

SOLVING THE QUANTUM LAYOUT PROBLEM FOR NISQ-ERA QUANTUM COMPUTERS VIA METAHEURISTIC ALGORITHMS

THESIS

Brian D. Curran Jr., BS
Second Lieutenant, USAF

Committee Membership:

Laurence D. Merkle, Ph.D.
Chair
David E. Weeks, Ph.D.
Member
Lt Col Patrick J. Sweeney, Ph.D.
Member

Abstract

In the noisy intermediate-scale quantum (NISQ)-era, quantum computers (QC) are highly prone to noise-related errors and suffer from limited connectivity between their physical qubits. Circuit transformations must be made to abstract circuits to address the noise and hardware constraints of NISQ-era devices. Such transformations introduce additional gates to the original circuit, thereby reducing the circuit's overall fidelity. To address the aforementioned constraints of NISQ-era QCs, dynamic remapping procedures permute logical qubits about physical qubits of the device to increase the fidelity of operations and make operations hardware-compliant. The quantum layout problem (QLP) is the problem of mapping logical qubits of the circuit to physical qubits of the target QC in a way that maximizes circuit fidelity and satisfies all device connectivity constraints. This research effort seeks to use metaheuristic algorithms to find high-quality solutions to the QLP. In this work, the QLP is mathematically modeled, integrated into various optimization algorithm domains, and resultant algorithms evaluated for efficiency and effectiveness. Moreover, fitness landscape analysis is performed based on the devised representation, objective functions, and search operators.

Acknowledgements

I would first like to thank my advisor, Dr. Laurence D. Merkle, for his feedback and guidance through this thesis. His critiques and comments throughout this process have helped me take a mere idea and craft a scientific body of work. Next, I would like to thank my committee members, Lt Col Patrick Sweeney and Dr. David Weeks, for their time spent reviewing and evaluating this thesis.

I would also like to thank my parents for all the encouragement they have given me over the years. They have taught me to never stop pursuing something you love, and to keep going when it gets tough.

Last, I would like to thank my loving fiancée. On my worst days, she always knows how to raise my spirits. Her patience and support throughout this journey are appreciated beyond words.

Lt Brian D. Curran Jr.

Table of Contents

List of Figures

List of Tables

List of Algorithms

SOLVING THE QUANTUM LAYOUT PROBLEM FOR NISQ-ERA QUANTUM COMPUTERS VIA METAHEURISTIC ALGORITHMS

I. Introduction

1.1 Motivation

A new, potentially disruptive, revolution in computing is underway: quantum computing. Quantum computers (QC) use the quantum physical phenomena of superposition and entanglement to perform computations that are infeasible on classical computers. According to Herman, countries such as China invest billions of dollars each year into QC technologies [1]. Research in quantum computing technologies is of utmost importance, as information security is at risk should this new form of computation mature to solve more complex problems. Moreover, mature QCs have the ability to solve several NP, NP-Complete, and NP-Hard problems currently intractable on classical computers. Consider Shor's algorithm, which finds the prime factors of a number in polynomial time. The factoring problem lies in the class NP, but no poly-time algorithms are known for classical computers. Thus, QCs appear to be able to tractably solve computationally challenging problems that their classical counterparts cannot.

While QCs exhibit substantial capabilities, state-of-the-art (SOTA) QCs cannot execute even moderately sophisticated programs due the unreliability of current quantum bits (qubits) and quantum gates applied to qubits, as well as the topological constraints of quantum hardware. Even though QCs are fickle devices, efforts have been made to mitigate noise-related issues via intelligent transpilation of quantum circuits. Specifically, when transpiling a circuit, numerous optimization passes are applied to a circuit that aim to maximize the probability of successful circuit execution on a target QC.

This research effort seeks to find good approximate solutions to a specific transpilation step applied to quantum circuits in which a logical quantum circuit is mapped onto a physical QC. This problem is commonly called the *Quantum Layout Problem* (QLP). Metaheuristics are an

acceptable approach to finding good solutions to the QLP due to the NP-hardness of the QLP and difficulty of relevant problem-instances. Moreover, previous work in this area of research mainly used greedy local (informed) search algorithms to find solutions to the QLP. In their strategies, algorithmic steps proceed based on locally available information about the circuit (e.g. upcoming gates that need to be executed). Metaheuristics can be used to instead consider global information about the entire circuit in its search process, an area of research on the QLP not well studied.

The potential impact of the results of this study are information superiority over adversaries. The utility of QCs is still unknown, but future research in this field is of highest military and civilian importance as the Pearl Harbor attack of the 21st century could very well be in the form of a cyberattack based on quantum technology. AFIT's mission is to "[e]ducate our Total Force military and civilian defense professionals to innovatively accomplish the deterrence and warfighting missions of the U.S. Air and Space Forces … today and tomorrow" ("Air Force Institute of Technology") [2]. This research fits directly into this mission, as attaining an understanding of these new quantum technologies is necessary for both prevention of and deterrence from attack by adversaries.

1.2 Problem Background

At this point in time, computer scientists can easily devise and implement algorithms on classical computers without knowledge of the low-level mechanisms of a classical computer. For instance, programmers commonly write code in a high-level programming language and compile their code into an executable program via a compiler. A programmer does not need to understand the bit-level realization of their high-level program since the compiler translates their program into a low-level executable. In contrast, current QCs require quantum programmers to write their code at the qubit-level. That is, quantum programmers must specify their algorithms to perform elementary operations on a collection of qubits. Currently, sufficient resources do no not exist for

quantum programmers to write high-level quantum programs and compile their programs to an executable for a QC.

Even though quantum programmers must write code at the qubit-level, transpilation steps must still be taken on their circuit in order for it to be executable on a QC. This is due to several constraints of QCs. First, abstract quantum circuits generally contain gate operations not realizable on QCs. Thus, one transpilation step converts the gates of the quantum circuit into functionally equivalent gates (or sequences of gates) that the QC understands. Before addressing the next constraint, note that the logical qubits of a quantum circuit must be placed in physical qubits of the target QC in order for operations to execute. This is like the situation in classical computing in which bits must be loaded into registers in order for logic gates to execute.

The second constraint is that QCs require all logical qubits q_α and q_β involved in a two-qubit gate operation be placed in physically adjacent physical qubits P_γ and P_δ, respectively. This is because when two qubits interact on the transmon QC architectures studied in this research effort, they must share a microwave-pulse wave-guide. If q_α and q_β are placed in non-adjacent physical qubits, the operation cannot execute since a microwave-pulse cannot be applied to both qubits concurrently. These constraints are commonly referred to as C_{NOT} constraints. Thus, another transpilation step ensures all logical qubit pairs (q_α, q_β) involved in two-qubit gate operations are placed in physical qubits (P_γ, P_δ) where P_γ and P_δ share a wave-guide.

A third constraint is that QCs are error-prone. Though gate operations on classical computers can fail, this occurs infrequently [3]. In addition, classical computers implement error-correcting codes to fix errors when they do occur. However, operations on QCs have much higher error rates. According to Tannu and Qureshi, single-qubit gate operations have a failure-rate of order 10^{-3}, while two-qubit gate operations have a failure-rate of order 10^{-2} [4]. Moreover, QCs are too resource-constrained to implement quantum-error-correcting codes. On top of that, noisy-

intermediate-scale quantum (NISQ) computers, the current SOTA quantum computing devices, have high variability in their gate-operation error rates. On a given day, physical qubits P_γ and P_δ may share a wave-guide that successfully performs a two-qubit operation with probability 99%. Then, on another day, that same wave-guide may only succeed with probability 85%. As such, during transpilation, it is essential to place logical qubits in physical qubits that are highly reliable, and that share wave-guides that are highly reliable. Additionally, QCs are constrained by the fact that qubits can only hold their quantum state for a given period of time before they lose their coherence. This limits the number of gate operations that can be performed in a quantum circuit.

Of the three aforementioned constraints of QCs, the latter two are the responsibilities of a QLP-solver. A QLP-solver seeks to place logical qubits in physical qubits such that all logical qubits involved in two-qubit operations are placed in adjacent physical qubits, and also attempts to place logical qubits in physical qubits that are highly reliable and share highly reliable wave-guides. These two objectives jointly constitute the QLP. The QLP addresses the question *what is the best mapping between logical and physical qubits that satisfies all C_{NOT} constraints and maximizes the probability that the circuit executes successfully?*

In general, no single mapping between physical and logical qubits (P2L mapping) satisfies all C_{NOT} constraints. As such, the QLP involves finding a collection of P2L mappings that collectively satisfy all C_{NOT} constraints. Then, (abstract) SWAP gates are applied to permute the logical qubits about the physical qubits to rearrange the mappings. Thus, a QLP-solver is also responsible for inserting SWAP gates to make the circuit executable. Since a SWAP gate is a two-qubit gate with a high associated error rate, inserting many SWAP gates degrades overall circuit fidelity (thereby affecting the second objective).

1.3 Research Objectives

Prior research on the QLP, such as work done by IBM research and various authors, predominantly use greedy local (informed) search algorithms to find solutions to the QLP (mainly the A^* search algorithm). Though their approaches vary slightly in their search algorithms, most use cost functions that attempt to minimize the number of additional gates added to the circuit by their QLP-solver or maximize the probability that all gates in the transpiled circuit execute successfully. The key point is their approaches greedily (albeit some with lookahead capability) choose steps to find locally optimal solutions to the QLP. However, choosing steps greedily for the QLP does not guarantee obtaining a circuit that globally minimizes (or maximizes) the objective. This research effort seeks to use metaheuristics that consider global information about the circuit (as opposed to local information used by previous research) to find higher-quality solutions of the QLP. The following research questions arise when considering the primary goals of this research:

1. How effective and efficient are various metaheuristic algorithms at finding high-quality solutions to the QLP?
2. For various QLP problem-instances, how can the topology of the fitness landscape induced by the representation, objective function(s), and search operator(s) devised for the metaheuristics-based (meta-based) QLP-solvers be characterized?

The first research question seeks to fill a void in an area of QLP research not extensively studied. As previously mentioned, prior research efforts (and now SOTA QLP-solvers) seek to find solutions to the QLP mainly via greedy heuristic approaches. Their approaches are limited by near-term knowledge about upcoming gates in the circuit, making decisions to solve QLP problem-instances based on heuristic cost functions that attempt to minimize or maximize global objectives. Research question #1 challenges this approach and asks whether global optimization

strategies based on metaheuristic algorithms can effectively and efficiently prune the search space of the QLP to find high-quality solutions.

The second research question seeks to gain an understanding of the fitness landscape of the QLP. In search algorithm development, the interplay between the representation, objective function, and search operators in the induced fitness landscape provide insight to tailor and choose effective algorithms to employ to find high-quality solutions. Wolpert and Macready introduce the no-free-lunch (NFL) theorem, which states "if some algorithm a_1's performance is superior to that of another algorithm a_2 over some set of optimization problems, then the reverse must be true over the set of all other optimization problems" [5]. In other words, the NFL theorem says that no optimization algorithm is superior to all other optimization algorithms over the set of all optimization problems. A good way to find effective optimization algorithms for a given problem is to empirically study global features of the search landscape and local features visible to the search operator(s). Regardless of the findings for research question #1, findings for the second research question are anticipated to identify features of the QLP fitness landscape that future researchers can exploit to devise novel optimization algorithms for the QLP.

1.4 Limitations and Assumptions

This research seeks to optimize the QLP-solver involved in quantum program transpilation for IBM transmon QC architectures. While there exist other variants of quantum computing architectures, such as quantum annealing-based QCs and topological QCs, this research focuses only on transmon architectures, targeting IBM's QCs specifically. For transmon architectures, this research assumes limited connectivity between physical qubits and noisy qubits/gate operations. Care should be taken in generalizing the results and conclusions of this research if connectivity is substantially increased and/or qubit noise is greatly reduced.

Next, the following additional assumptions are made in this research effort:

- For a quantum circuit to execute successfully, all 1-qubit and 2-qubit gates must execute correctly. Furthermore, these events are assumed to be independent.
- Qubit coherence errors are correlated with circuit depth. The longer a quantum circuit executes, the higher the probability that a coherence error occurs.

In addition, the devised software for the meta-based QLP-solvers is intended to be integrated into the Qiskit quantum computing framework. As such, most of the software is written in the Python programming language. Therefore, running the software requires a user to have Python and Qiskit installed. Python 3.7.4 is used. Table 1 provides the main Python libraries used in this research effort.

Name	Version
Qiskit	0.23.1
Qiskit-Terra	0.16.1
Qiskit-Aer	0.7.1
Qiskit-Ignis	0.5.1
Qiskit-IBMQ-Provider	0.11.1
Qiskit-Aqua	0.8.1
Networkx	2.3
Json	2.0.9
Pymoo	0.4.2.1
Numpy	1.19.2

Table 1: Main Python libraries used in this research effort.

Finally, all fitness landscape analysis is built upon the proposed representations, objective functions, and search operators. Thus, the findings for research question #2 are only applicable under the imposed fitness landscape. Alternative representations, objective functions, and search operators induce alternative fitness landscapes, which may have vastly different features than those identified in this research effort. Nonetheless, techniques employed in this research can be augmented to explore alternative fitness landscapes.

1.5 Mathematical Notation and Conventions

In this section, mathematical notation and conventions used in the remainder of this thesis are defined. Table 2 provides descriptions of various notation.

Notation	Description
$[a, b]$	The set of all reals between a and b, inclusive.
$[\![a, b]\!]$	The set of all integers between a and b, inclusive.
$y = (1,2,3, \dots)$	y is a sequence of elements.
$\lvert y \rvert$	The number of elements in sequence y.
y_i $y^* = (a_0, b_1, a_1, b_1, \dots)$ $a_i \in y^*$	Element of sequence y with index i (indexing is 0-based). Against common practice, given a sequence y^* composed of objects of types a, b, etc., let $a_i \in y^*$ be the set of all objects of type a from sequence y^*.
$\{0,1\}^n$	The set of strings of length n with each character being either zero or one.
$G = (V, E)$	A graph G with vertex-set V and edge-set E. The vertex-set and edge-set of G are also commonly denoted $V(G)$ and $E(G)$, respectively.
$f = \begin{pmatrix} 0 & 1 \\ 1 & 2 \end{pmatrix} \dots$ $i \mapsto j$	A bijective function f that maps a domain to a codomain. In this example, $f(0) = 1$ and $f(1) = 2$. Element i of the domain maps to element j of the codomain. In function f, $0 \mapsto 1$.

Table 2: Notation used in this thesis.

Given a graph G and a path p from vertex $V_i \in E(G)$ to $V_j \in E(G)$, p is a sequence of vertices where $p_i \in V(G)$. The length of path p is the number of vertices in the path and is denoted $|p|$. For a path of length k from V_i to V_j, $p_0 = V_i$, $p_{k-1} = V_j$, and $p_1 \dots p_{k-2}$ are intermediate vertices between V_i and V_j.

A list is a computer science data structure similar to a sequence in mathematics. A list l is composed of elements, each of which is addressable by index (0-based indexing). For example, given a list $l = [a, b, c, d]$, $l[0] = a$ and $l[2] = c$. The number of elements in a list is denoted by

$|l|$. A list has an associated function $append(i)$ which takes an element i and adds it to the end of the list. In the above example, $l.append(e)$ augments l as follows: $l = [a, b, c, d, e]$.

1.6 Document Overview

Chapter II provides background information on quantum computation, the transpilation process required to execute quantum circuits on quantum devices, IBM's quantum computing framework, and previous research on the QLP. In addition, metaheuristic algorithms and fitness landscapes are presented and defined. Chapter III defines the methodology used to design and analyze the proposed meta-based QLP-solvers. Moreover, fitness landscape definition and analysis procedures are defined in Chapter III. Chapter IV presents the effectiveness and efficiency of the proposed meta-based QLP-solvers compared against SOTA QLP-solvers, as well as an analysis of the fitness landscapes of the QLP. Finally, Chapter V concludes this work, addresses the research questions, and provides future directions for research in QLP-solver development.

II. Background and Literature Review

2.1 Overview

This chapter provides necessary background information on quantum computation and the main problem addressed in this research effort: The QLP. Section 2.2 presents, via analogy to classical computing concepts, the quantum analogues of bits, gates, and circuits. Section 2.3 explores the necessity of quantum program transpilation (QPT), defines the QLP, and presents SOTA QLP-solvers. Section 2.4 presents a widely used quantum computing framework developed by IBM. Section 2.5 then provides necessary background information on the metaheuristic algorithms used in this research effort to find solutions to the QLP. Section 2.6 explains fitness landscapes and provides techniques to analyze them. Finally, Section 2.7 presents a problem related to the QLP called the token-swapping problem.

2.2 Quantum Computation

Quantum computation is commonly introduced by analogy to classical computation. The world of classical computation mainly involves bits, logic gates, and logic circuits. Logic gates operate on bits to produce new states of the bits, and logic circuits are a collection of logic gates that operate on a set of bits. In Sections 2.2.1 through 2.2.3, the quantum analogues of the bit, gate, and circuit are introduced, respectively.

2.2.1 Qubits

In classical computation, bits are the basic unit of information. The mathematical object of a bit represents a logical state with one of two possible values: 0 or 1. In quantum computation, qubits are the basic unit of information. Two possible states of a qubit are $|0\rangle$ and $|1\rangle$, which correspond to the states 0 and 1 of a classical bit, respectively. Qubits can also be in states other than $|0\rangle$ and $|1\rangle$. Qubits can be in a linear combination of states, called *superpositions* [7]. Let $|\psi\rangle$ represent an arbitrary state of a qubit.

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle : \alpha, \beta \in \mathbb{C} \wedge |\alpha|^2 + |\beta|^2 = 1 \quad (2.1)$$

Therefore, while a classical bit's state is a discrete number i such that $i \in \{0,1\}$, a qubit's state is a vector in a two-dimensional complex vector space. The states $|0\rangle$ and $|1\rangle$ are known as computational basis states and form an orthonormal basis over the vector space [7]. In Equation 2.1, α and β are amplitudes where the absolute square of their values represent the "$|0\rangle$-ness" and "$|1\rangle$-ness" of a qubit's state, respectively.

A classical bit's value can be examined via memory access. That is, at any point in time, the exact state of a classical bit can be determined. However, the state of a qubit cannot be examined to determine its state (i.e. the values of α and β). Instead, when a qubit is examined, the state $|0\rangle$ is obtained with probability $|\alpha|^2$ and the state $|1\rangle$ with probability $|\beta|^2$. This strange behavior is due to one of the postulates of quantum mechanics (QM). Thus, classical bits are governed by the laws of classical mechanics, while qubits are governed by the laws of QM. In Equation 2.1, since the absolute squares of α and β equate to probabilities, the sum of the absolute squares must equal 1 (thus imposing the need for the normalization constraints in the definition of $|\psi\rangle$) [8].

Consider a qubit in the state $|\psi_0\rangle = 1 \cdot |0\rangle + 0 \cdot |1\rangle$. First, this is a valid state as $|1|^2 + |0|^2 = 1 \wedge 0,1 \in \mathbb{C}$. Reduction of the qubit's state results in $|\psi_0\rangle = |0\rangle$. Thus, this qubit is in the pure ground state. Measurement of this qubit results in $|0\rangle$ 100% of the time. An example of a qubit in the pure excited state is $|\psi_1\rangle = 0 \cdot |0\rangle + 1 \cdot |1\rangle = |1\rangle$. Next, consider a qubit in a superposition state: $|\psi_+\rangle = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$. This is also a valid state as $\left|\frac{1}{\sqrt{2}}\right|^2 + \left|\frac{1}{\sqrt{2}}\right|^2 = 1 \wedge \frac{1}{\sqrt{2}} \in \mathbb{C}$. Since the absolute squares of α and β represent the "$|0\rangle$-ness" and "$|1\rangle$-ness" of the qubit's state, measurement of this qubit results in $|0\rangle$ with probability $\left|\frac{1}{\sqrt{2}}\right|^2 = 50\%$ and $|1\rangle$ with probability $\left|\frac{1}{\sqrt{2}}\right|^2 = 50\%$. Before measurement, the qubit's state was neither purely $|0\rangle$ nor $|1\rangle$, but a mixture

of both. After measurement, however, the qubit's state jumps to one of the computational basis states with probabilities defined by α and β.

Since a quantum state must satisfy the normalization constraint $|\alpha|^2 + |\beta|^2 = 1$, Equation 2.1 can be rewritten as follows [7]:

$$|\psi\rangle = e^{i\gamma}(\cos\frac{\theta}{2}|0\rangle + e^{i\varphi}\sin\frac{\theta}{2}|1\rangle) : \theta, \varphi, \gamma \in \mathbb{R} \tag{2.2}$$

Moreover, the factor of $e^{i\gamma}$ in Equation 2.2 can be discarded as it has no observable effects according to Nielsen and Chuang [7]. Therefore, a qubit's state can be expressed as follows:

$$|\psi\rangle = \cos\frac{\theta}{2}|0\rangle + e^{i\varphi}\sin\frac{\theta}{2}|1\rangle : \theta, \varphi \in \mathbb{R} \tag{2.3}$$

By Equation 2.3, the state of a single qubit can be expressed via the polar coordinates θ and φ, where θ and φ define a point on a 3-dimensional unit sphere [7]. This sphere is commonly referred to as the *Bloch sphere*. Figure 1 provides an illustration of a Bloch sphere.

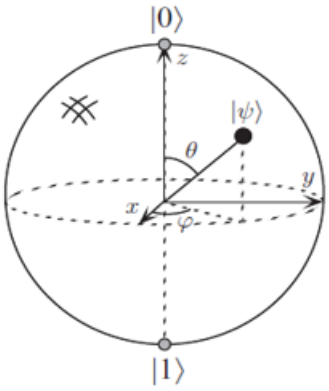


Figure 1: A Bloch sphere, used to visualize the state of a single qubit. Reproduced from Nielsen and Chuang [7].

Each point on the Bloch sphere represents a unique state of a single qubit. The antipodal points correspond to the orthogonal computation basis states $|0\rangle$ and $|1\rangle$. Also notice that the state-space

of a single qubit is infinite, as illustrated by the infinite number of points on the surface of the Bloch sphere. While the Bloch sphere is a useful concept to visualize the state of a single qubit, no generalization of the Bloch sphere exists beyond a 2-level system (i.e. single qubit).

Multiple Qubit Systems: Prior to this point, only systems containing a single qubit have been defined and analyzed. To explore the concept of multiple qubit systems, a system of two classical bits is first analyzed. In a classical two bit system, the possible states of the system are 00, 01, 10, and 11. Correspondingly, a system of two qubits has four computational basis state: $|00\rangle, |01\rangle, |10\rangle$, and $|11\rangle$. A pair of qubits can also exist in superpositions of the four aforementioned basis states [7]. Thus, the state of a pair of qubits can be defined as follows:

$$|\psi\rangle = \alpha_{00}|00\rangle + \alpha_{01}|01\rangle + \alpha_{10}|10\rangle + \alpha_{11}|11\rangle \tag{2.4}$$

s.t.

$$\alpha_{00}, \alpha_{01}, \alpha_{10}, \alpha_{11} \in \mathbb{C} \wedge \sum_{x\in\{0,1\}^2} |\alpha_x|^2 = 1$$

In general, a system of n qubits has 2^n computation basis states, and the system can exist in superpositions of all computational basis states. As such, the state of an n qubit system can be defined as follows:

$$|\psi\rangle = \sum_{x\in\{0,1\}^n} \alpha_x|x\rangle \tag{2.5}$$

s.t.

$$\forall\alpha_x \in |\psi\rangle, \alpha_x \in \mathbb{C} \wedge \sum_{x\in\{0,1\}^n} |\alpha_x|^2 = 1$$

Entanglement: A strange quantum phenomenon can occur in multi-qubit systems called entanglement. To aid in understanding entanglement, consider the following two-qubit state:

$$|\psi\rangle = \frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|11\rangle.$$

This state is commonly referred to as a *Bell state* or an *EPR pair*. Consider measuring only the first qubit of the Bell state. Upon measurement of the first qubit, $|0\rangle$ is obtained with probability 50% and $|1\rangle$ with probability 50%. The state of the system after measuring the first qubit is either $|\psi'\rangle = |00\rangle$ (when $|0\rangle$ measured on first qubit) or $|\psi'\rangle = |11\rangle$ (when $|1\rangle$ measured on first qubit). Oddly, measurement of the second qubit always results in the same computational basis state measured on the first. These measurement outcomes are said to be *correlated* [7]. Even more strangely, the qubits continue to be correlated even if the distance between them is vast (e.g. one qubit is on Earth and the other is in a distant galaxy). That is, the state measured on the first qubit instantaneously "pops" the second qubit to the same state, regardless of the distance separating them. This appears to violate the universal speed limit (i.e. information cannot travel faster than the speed of light). Entanglement puzzled Albert Einstein, and he coined the phenomenon "spooky action at a distance" [7]. Later, physicist John Bell proved that correlation does not violate the laws of physics, and debunked alternative theories (e.g. hidden-variable theory). His work also proved that "the measurement correlations in the Bell state are stronger than could ever exist between classical systems" [7].

An entangled quantum state is one which cannot be expressed as the product of its constituent qubits. To illustrate this, consider the following non-entangled state:

$$|\psi_{\overline{entangled}}\rangle = \frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|01\rangle.$$

This state can be factored into the product of its constituent qubits as follows:

$$|\psi_{\overline{entangled}}\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |01\rangle) = \frac{1}{\sqrt{2}}(|0\rangle)(|0\rangle + |1\rangle).$$

Thus, the first qubit is in the state $|0\rangle$ and the second is in the superposition state $\frac{|0\rangle+|1\rangle}{\sqrt{2}}$. Now consider factoring the entangled Bell State:

$$|\psi_{entangled}\rangle = \frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|11\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle).$$

No further factoring can be done to this state. Therefore, the state of the first and second qubits cannot be expressed individually. Thus, their states are entangled.

2.2.2 Quantum Gates

Without loss of generality, quantum gates can be divided into three categories: single-qubit, two-qubit, and measurement gates. This is because all multi-qubit gates with more than two input qubits can be decomposed into sequences of single- and two-qubit gates [9]. In order to understand how a quantum gate manipulates the state of a qubit(s), the notation describing the state of a qubit must be explained. Previously, a given qubit's state was expressed via equations containing $|0\rangle$ and $|1\rangle$. The bracket-notation used in these equations is called Dirac notation. Dirac notation can equivalently be expressed in vector notation. For example, a single qubit's state can be expressed in vector notation as follows:

$$\alpha|0\rangle + \beta|1\rangle = \begin{pmatrix}\alpha\\ \beta\end{pmatrix} \tag{2.6}$$

In general, a multi-qubit state of n qubits can be expressed in vector notation as follows:

$$\sum_{x\in\{0,1\}^n} a_x|x\rangle = \begin{pmatrix}\alpha_0\\ \alpha_1\\ \cdots\\ \alpha_{2^n-1}\end{pmatrix} \tag{2.7}$$

In Dirac notation, the state of a system of n qubits is typically defined as $|\psi\rangle = |Q_0Q_1 \ldots Q_{n-1}\rangle$. In vector notation, this equates to the tensor product of each $Q_i \in |\psi\rangle$ (from left to right). Thus,

$$|\psi\rangle = |Q_0Q_1 \ldots Q_{n-1}\rangle = |Q_0\rangle \otimes |Q_1\rangle \otimes \ldots \otimes |Q_{n-1}\rangle = \begin{pmatrix}\alpha_{0,0}\\ \alpha_{0,1}\end{pmatrix} \otimes \begin{pmatrix}\alpha_{1,0}\\ \alpha_{1,1}\end{pmatrix} \otimes \ldots \otimes \begin{pmatrix}\alpha_{n-1,0}\\ \alpha_{n-1,1}\end{pmatrix},$$

where α_{ij} is the amplitude on $|j\rangle$ of qubit Q_i.

All valid quantum gates must be both linear and norm-preserving. The linearity requirement is due to the postulates of QM [7]. Quantum gates must also be norm-preserving in order to create transformations of the quantum state which still satisfy the normalization constraint of $|\psi\rangle$. Given a quantum gate U, U is a valid gate if $U^{\dagger}U = UU^{\dagger} = I$ [7]. Therefore, a valid quantum gate is always be defined by a unitary matrix.

Single-Qubit Gates: Operations on a single-qubit are described by norm-preserving 2×2 unitary matrices. To understand the workings of single-qubit gates, consider a classical NOT gate. A classical NOT gate operates on a single bit and its actions are described by the following truth table:

Input (a)	**Output ($\sim a$)**
0	1
1	0

Classical NOT gate truth table.

Now, a quantum analogue of a NOT gate should take the computation basis states $|0\rangle \rightarrow |1\rangle$ and $|1\rangle \rightarrow |0\rangle$. In addition, a quantum NOT gate should negate any superposition state appropriately. Due to the linearity of QM, an operator which defines how the basis states transform also defines how any superposition (linear combination) state transforms [7]. Thus, to create any single-qubit gate, define a 2×2 unitary matrix where the first column defines the resultant state of the operator applied to $|0\rangle$ and the second column defines the resultant state of the operator applied to $|1\rangle$. Let X define a quantum NOT gate.

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

In the proposed quantum NOT gate above, the first column maps $|0\rangle = \binom{1}{0} \mapsto \binom{0}{1} = |1\rangle$, and the second column maps $|1\rangle = \binom{0}{1} \mapsto \binom{1}{0} = |0\rangle$. Further, X is a unitary matrix because $X^\dagger X = XX^\dagger = I$. Therefore, X is a valid quantum gate and transforms a qubit's state by rotating it by π radians about the x-axis of the Bloch sphere [10]. Consider the following examples which illustrate the X gate applied to various quantum states:

$$X|0\rangle = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}\binom{1}{0} = \binom{0}{1} = |1\rangle$$
$$X|1\rangle = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}\binom{0}{1} = \binom{1}{0} = |0\rangle$$

Another important single-qubit gate is known as the "Hadamard gate" (H). This gate transforms the state $|0\rangle \mapsto |+\rangle$ and $|1\rangle \mapsto |-\rangle$, where $|+\rangle = \frac{1}{\sqrt{2}}\binom{1}{1}$ and $|-\rangle = \frac{1}{\sqrt{2}}\binom{1}{-1}$. The H gate, based on how it transforms the basis states, is defined as follows:

$$H = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}.$$

The provided H gate is a valid quantum operation because $H^\dagger H = HH^\dagger = I$. The H gate transforms a qubit's state by rotating it by π radians about the z-axis followed by $\frac{\pi}{2}$ radians about the y-axis of the Bloch sphere [10].

Two-Qubit Gates: Operations on two qubits are described by norm-preserving 4×4 unitary matrices. To understand how a two-qubit gate works, consider a classical XOR gate. A classical XOR gates operates on two bits and its actions are described by the following truth table:

Input 1 (a)	**Input 2 (b)**	**Output ($a \oplus b$)**
0	0	0
0	1	1
1	0	1
1	1	0

Classical XOR gate truth table.

Now, a quantum analogue of the XOR gate should take the computational basis states $|00\rangle \mapsto |00\rangle, |01\rangle \mapsto |01\rangle, |10\rangle \mapsto |11\rangle$, and $|11\rangle \mapsto |10\rangle$. Here, the first qubit is the control and the second is the target (thus, the resultant state should preserve the state of the control qubit and flip the state of the target qubit if the control is $|1\rangle$). Similar to single-qubit gates, to create a two-qubit gate define a 4×4 unitary matrix where the first column defines the resultant state of the operator applied to $|00\rangle$, …, and the final column defines the resultant state of the operator applied to $|11\rangle$. Let C_{NOT} define a quantum XOR gate.

$$C_{NOT} = \begin{pmatrix} I_{2\times2} & 0_{2\times2} \\ 0_{2\times2} & X \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}.$$

In the provided C_{NOT} gate, the first column maps $|00\rangle = (1\ 0\ 0\ 0)^T \mapsto (1\ 0\ 0\ 0)^T = |00\rangle$, the second column maps $|01\rangle = (0\ 1\ 0\ 0)^T \mapsto (0\ 1\ 0\ 0)^T = |01\rangle$, the third column maps $|10\rangle = (0\ 0\ 1\ 0)^T \mapsto (0\ 0\ 0\ 1)^T = |11\rangle$, and the fourth column maps $|11\rangle = (0\ 0\ 0\ 1)^T \mapsto (0\ 0\ 1\ 0)^T = |10\rangle$. Further, C_{NOT} is a unitary matrix because $C_{NOT}^{\dagger} C_{NOT} = C_{NOT} C_{NOT}^{\dagger} = I$. A generalization of the C_{NOT} gate is a controlled-u gate (C_u). A C_u gate can be defined as follows:

$$C_u = \begin{pmatrix} I_{2\times2} & 0_{2\times2} \\ 0_{2\times2} & u \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & u_{0,0} & u_{0,1} \\ 0 & 0 & u_{1,0} & u_{1,1} \end{pmatrix}.$$

The C_u gate functions similar to the C_{NOT} gate, except when the control qubit is in the state $|1\rangle$, unitary u is applied to the target (where u is a 2×2 unitary matrix).

One final 2-qubit gate that is of importance to this research effort is the $SWAP$ gate. The $SWAP$ gate exchanges the state of two qubits. The $SWAP$ gate is defined as follows:

$$SWAP = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

The $SWAP$ gate is best illustrated via example. Consider the following examples:

$$SWAP|01\rangle = |10\rangle$$
$$SWAP|10\rangle = |01\rangle$$
$$SWAP|+-\rangle = |-+\rangle$$

Thus, the $SWAP$ gate exchanges the states of the two qubits which it acts on. This will become particularly useful in quantum transpilation. In this section, only a few gates are presented. Figure 2 provides some commonly used 1- and 2-qubit gates and their associated unitaries.

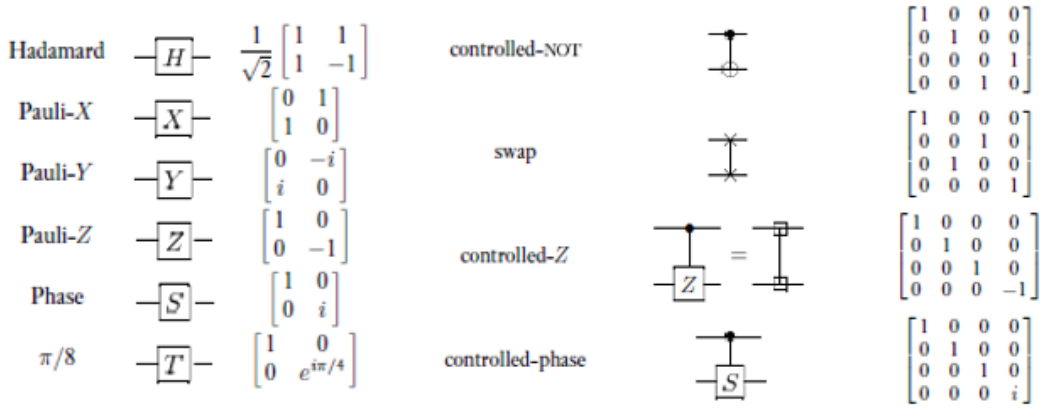


Figure 2: Commonly used 1- and 2-qubit gates. Reproduced from Nielsen and Chuang [7].

Measurement Gates: The only valid non-unitary gate that can be applied to qubits is the measurement gate. A measurement gate collapses the wave function defining a qubit's state, and is an irreversible operation as consequence. As an example, consider a single qubit prepared in the state $|\psi\rangle = |+\rangle = \begin{pmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{pmatrix}$. Measurement can be performed in any orthonormal basis [7], however for this research, understanding measurement in the computational basis is sufficient. Let M_c denote a measurement gate which projects a quantum state into the computational basis. Applying the measurement gate $M_c|+\rangle$ results in a measurement of $|0\rangle$ 50% (post-measurement

$|\psi\rangle = |0\rangle$) of the time (post-measurement, $|\psi\rangle = |0\rangle$) and $|1\rangle$ 50% of the time (post-measurement, $|\psi\rangle = |1\rangle$).

2.2.3 Quantum Circuits

A quantum circuit can be defined as a sequence of c unitary operations $O = (U_0, U_1, \dots, U_{c-1})$ to apply to system of n qubits defined by $|\psi\rangle = |q_0 q_1 \dots q_{n-1}\rangle$. In this definition, $O_0, O_1, \dots, O_{c-1}$ are $2^n \times 2^n$ dimensional unitary matrices. The resulting state $|\psi'\rangle$ of applying all unitary operations to $|\psi\rangle$ can be defined as follows:

$$|\psi'\rangle = \prod_{k=1}^{c} O_{c-k}|\psi\rangle \tag{2.8}$$

This concept is best understood via an example. Consider the quantum circuit in Figure 3.

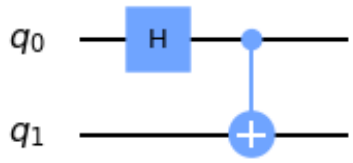


Figure 3: Example quantum circuit which creates a Bell state.

Preliminaries:

1. When a quantum circuit is drawn, assume that each qubit is initialized to the state $|0\rangle$. In Figure 3, this means $|q_0\rangle = |q_1\rangle = |0\rangle$.
2. A circuit can be divided into *layers*. A layer l_i is defined as "contain[ing] only gates that act on distinct sets of qubits" [11]. This will be important for circuit analysis. Figure 4 shows the circuit in Figure 3 divided into layers.

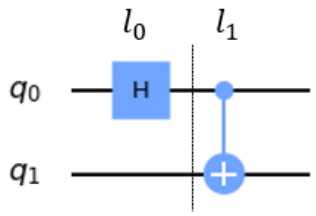


Figure 4: Circuit in Figure 3 divided into layers l_0 and l_1.

For each layer l_i of a circuit, if a gate is not applied to qubit q_j, this implies that the identity gate I is being applied to q_j.

3. Each layer l_i of a circuit can be defined by a $2^n \times 2^n$ unitary U_i which is equal to the tensor product, from top to bottom, of the gates within it.
4. Equation 2.8 defines the resultant state of applying all gates in the circuit to the system of qubits.

To process these preliminaries, first recognize that this is a system of 2 qubits defined by initial state $|\psi_0\rangle = |00\rangle$ (by preliminary 1). Next, after dividing the circuit into layers l_0 and l_1, preliminary 3 specifies that since no gate is applied to q_1 in l_0, this implies that the identity gate is implicitly being applied to q_1 in l_0. The identity gate (i.e. the "do-nothing-but-make-the-math-work" gate) is defined by the following unitary:

$$I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}.$$

Thus, the identity gate can be applied to q_1 in l_0 and does not alter the results of the computation. Finally, preliminary 4 says that each layer l_i can be expressed as a unitary U_i. Figure 5 shows the circuit divided into layers and the associated unitaries for each layer.

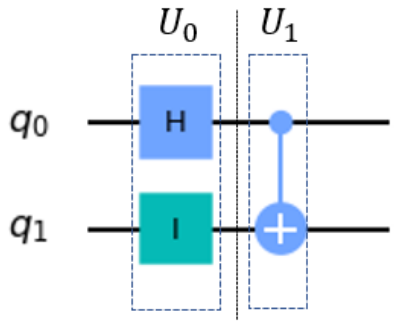


Figure 5: Circuit from Figure 3 divided into layers and associated unitaries.

Based on preliminary 4, the unitaries U_i are calculated as the tensor product, from top to bottom, of all gates in l_i. For this circuit, $U_0 = (H \otimes I)$ and $U_1 = C_{NOT}$. Finally, analysis of this circuit should confirm that the resultant state-vector is $|\psi'\rangle = \frac{|00\rangle+|11\rangle}{\sqrt{2}}$.

$$
\begin{aligned}
|\psi'\rangle &= \prod_{k=1}^{c} U_{c-k}|\psi\rangle \\
&= (U_1 \cdot U_0)|\psi\rangle \\
&= [(C_{NOT}) \cdot (H \otimes I)]|00\rangle \\
&= \left[\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \cdot \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \end{pmatrix}\right] \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \\
&= \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \end{pmatrix} = \frac{|00\rangle + |11\rangle}{\sqrt{2}}
\end{aligned}
$$

This process can be applied to any arbitrary quantum circuit to analyze how it operates. However, some situations can occur that make this procedure cumbersome. For instance, consider the circuit in Figure 6.

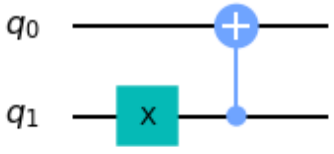


Figure 6: A circuit which is non-trivial to analyze.

This circuit can clearly be divided into layers l_0 and l_1. The unitary associated with l_0 is $U_0 = (I \otimes X)$. However, what is the unitary associated with layer l_1? At first glance, it appears that $U_1 = C_{NOT}$. However, the C_{NOT} unitary expects the first qubit to be the control and the second to be the target. To get around this issue, the following circuit identity is used:

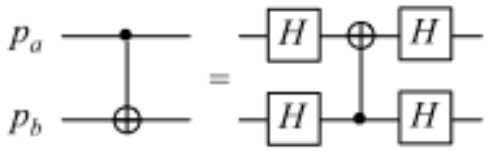


Figure 7: Reversal circuit identity for backwards C_{NOT} operation. Reproduced from Murali et al. [12].

Thus, an equivalent circuit, incorporating the reversal identity, is shown in Figure 8. This circuit can now be analyzed using Equation 2.8.

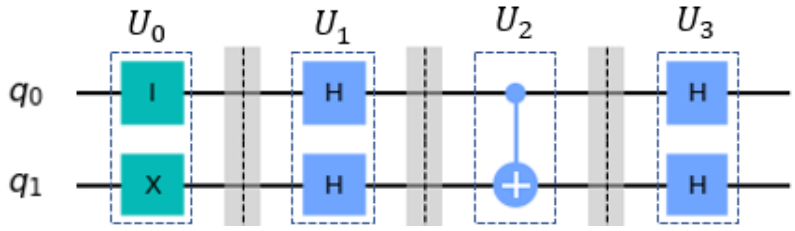


Figure 8: Equivalent circuit to Figure 6 which can be analyzed. Here, $U_0 = (I \otimes X)$, $U_1 = (H \otimes H)$, $U_2 = C_{NOT}$, and $U_3 = (H \otimes H)$.

Another situation is when the distance between the control and target qubits of a controlled unitary operation is greater than 1 (e.g. for a 3 qubit system, a C_{NOT} between control q_0 and target q_2). For this case, the generalized bridge operation defined in Section 3.3.1.3 or SWAP gates can be used. Figure 9 shows another circuit identity which, along with the reversal of Figure 7, is important for quantum transpilation. Figure 9 shows that a SWAP gate decomposes to three C_{NOT} gates. The second step in Figure 9 applies a reversal to the middle C_{NOT} operation.

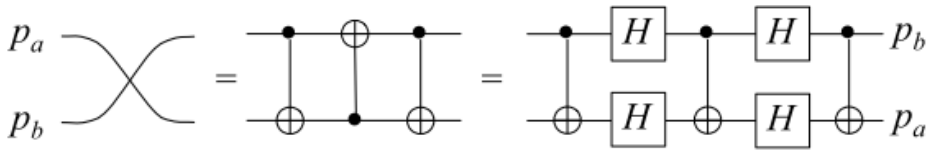


Figure 9: SWAP gate decomposition. Reproduced from Siraichi et al. [13].

Overall, while this method of analysis may appear to circumvent the need for QCs (i.e. why use a QC if basic linear algebra can be used to analyze a circuit), consider a system with $n = 500$ qubits. Then the unitaries defining each layer of a circuit for this system would be of size $2^{500} \times 2^{500}$. Performing arithmetic (or let alone persisting) such unitaries is computationally unfeasible for classical computers.

2.3 Quantum Program Transpilation

In this section, the need for and steps taken to transpile a quantum program are explored. Section 2.3.1 explains why QPT is necessary. Section 2.3.2 then explores a particularly difficult step of QPT and the focus of this research effort, the QLP. Finally, Section 2.3.3 presents SOTA optimization techniques used to find solutions to the QLP.

2.3.1 Necessity of Quantum Program Transpilation

Although an abstract quantum circuit consists of low-level operations which are performed on qubits, limitations of the backend QC require transformations be made to the input logical circuit. Without these transformations, the circuit cannot execute on the backend QC. In QPT, two steps are required to make a circuit executable on a given backend QC.

1. The gates used in the logical circuit must be converted into gates the backend QC understands.
2. $\forall(q_{control}, q_{target})$ pairs of logical qubits involved in C_{NOT} operations in the circuit, $q_{control}$ and q_{target} must be placed in adjacent physical qubits.

Gate Decomposition: The first step is necessary because the set of gates available to quantum programmers is typically larger than the set of gates the backend QC understands. A given backend QC has a basis set of gates. According to Adedoyin et al., for most of IBM's QCs, the basis set of gates is $\{u_1, u_2, u_3, I, C_{NOT}\}$ [9]. The u_1, u_2, and u_3 gates are parameterized single-qubit gates (shown in Figure 10 below). The basis gate-set must be a universal gate-set, meaning that any gate of an abstract quantum circuit can be realized by a sequence of the basis gates. Fortunately, IBM's QCs all have basis gate-sets which are universal. Thus, in the first step of QPT, the gates of the circuit are decomposed into functionally equivalent sequences of the basis gates.

$$u1(\lambda) = \begin{pmatrix} 1 & 0 \\ 0 & e^{\lambda i} \end{pmatrix},$$
$$u2(\phi, \lambda) = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & -e^{\lambda i} \\ e^{\phi i} & e^{(\lambda i + \phi i)} \end{pmatrix},$$
$$u3(\theta, \phi, \lambda) = \begin{pmatrix} \cos\frac{\theta}{2} & -e^{\lambda i}\sin\frac{\theta}{2} \\ e^{\phi i}\sin\frac{\theta}{2} & e^{(\lambda i + \phi i)}\cos\frac{\theta}{2} \end{pmatrix}$$

Figure 10: u_1, u_2 and u_3 gate definitions. Reproduced from Zhang et al. [14].

C_{NOT} Constraints: Next, a problem arises from the fact that logical qubit pairs involved in C_{NOT} operations must be placed in physically adjacent physical qubits. An abstract quantum circuit of M logical qubits assumes that a subgraph of the topology defining the backend QC is isomorphic to κ_M. However, in reality, the backend topology has limited connectivity between its physical qubits. In the literature, these constraints are commonly referred to as C_{NOT} *constraints* [11]. To illustrate this concept, consider the logical circuit and backend QC in Figure 11.

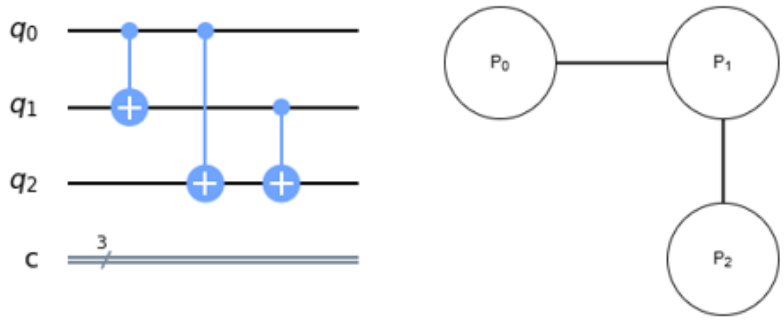


Figure 11: Logical circuit (left) to execute on backend QC (right).

Using the circuit of Figure 11, a coupling map Ω and related coupling graph G_Ω are constructed in Figure 12 to illustrate the interactions between logical qubits in the circuit, as done by Murali et al. [12]. For Ω, the element (q_i, q_j) is added if there is a C_{NOT} operation between q_i and q_j in the circuit. For G_Ω, add the directed edge (q_i, q_j) if $(q_i, q_j) \in \Omega$.

$$\Omega = \big((q_0, q_1), (q_0, q_2), (q_1, q_2)\big)$$

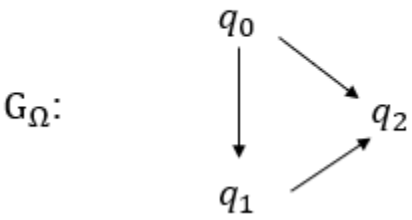


Figure 12: Coupling map and coupling graph for the circuit in Figure 11.

Figure 12 illustrates the fact that G_Ω is not isomorphic to a subgraph of the topology defining the backend QC from Figure 11. Therefore, there exists no placement of logical on physical qubits which satisfies all C_{NOT} constraints. In other words, there is no single mapping between logical and physical qubits which places all logical qubits involved in C_{NOT} operations in adjacent physical qubits. This turns out to be generally true for most conceivable circuits. That is, in general, no single mapping between logical and physical qubits satisfies all C_{NOT} constraints.

Addressing C_{NOT} Constraints: A quantum program transpiler is also responsible for making all C_{NOT} operations in the input circuit valid. That is, the transpiler must ensure that for each C_{NOT} operation in the circuit, the control and target qubits of the operation must be placed in adjacent physical qubits. Since no single mapping can generally satisfy all C_{NOT} constraints, a mechanism to change the mapping must be introduced. Mappings between logical and physical qubits can be changed via $SWAP$ operations. $SWAP$ operations permute the logical qubits along the physical qubits, thereby changing the mapping. $SWAP$ operations, like C_{NOT}s, can only be applied to physically adjacent qubits (for the curious reader, this is because a $SWAP$ decomposes into 3 C_{NOT} ops). Figure 13 illustrates steps that can be taken to address C_{NOT} constraints.

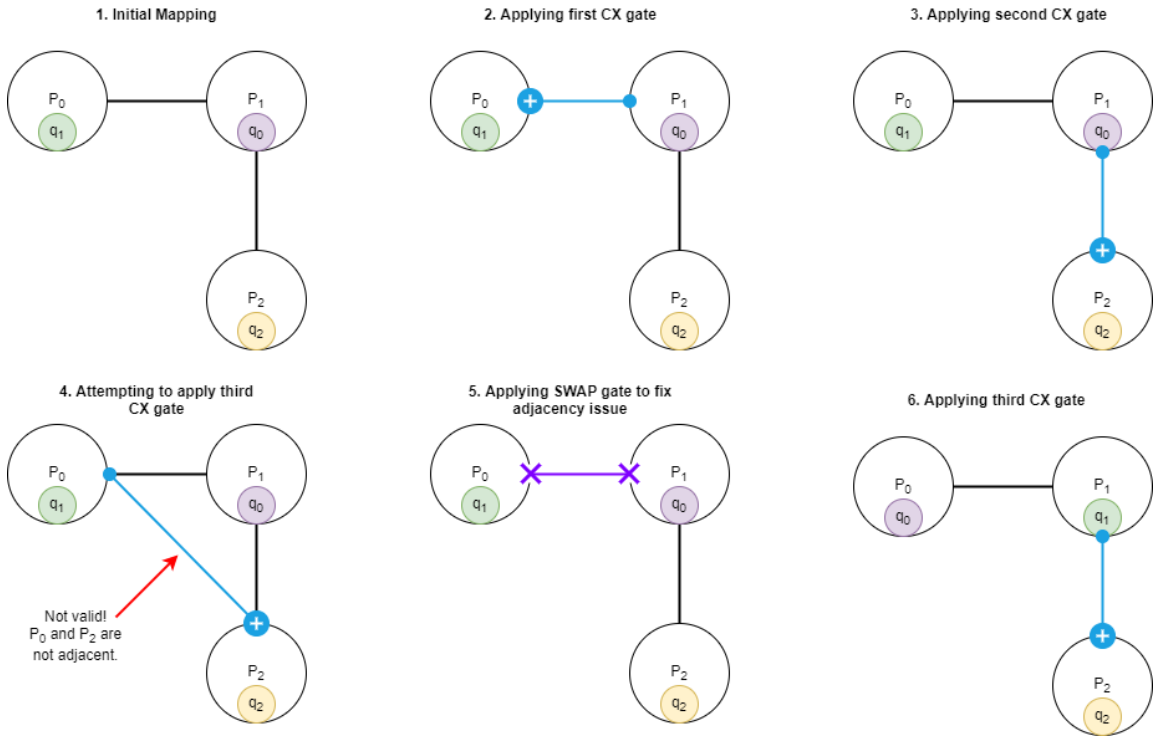


Figure 13: Steps to satisfy C_{NOT} constraints for the circuit, backend of Figure 11. In this figure, the logical circuit is superimposed onto the physical topology to illustrate that the operations are happening, in reality, between the physical qubits which hold each logical qubit.

In step 1, an initial mapping between logical and physical qubits is established. In this example, the mapping which allowed the most C_{NOT} operations to be executed was chosen.

Section 2.3.3 will discuss how SOTA transpilers determine this mapping. Next, the first gate to execute in the circuit is a C_{NOT} between (q_0, q_1). Since the initial mapping placed q_0 and q_1 in adjacent physical qubits, this operation can execute successfully (shown in step 2). The next gate is a C_{NOT} between (q_0, q_2). As in step 2, step 3 can execute successfully based on the initial mapping. In step 4, the application of the final C_{NOT} gate between (q_1, q_2) is attempted. However, since the physical qubits holding q_1 and q_2 are not adjacent, this gate cannot be executed. In step 5, the mapping between logical and physical qubits is changed via a $SWAP$ operation in order to make the physical qubits holding q_1 and q_2 adjacent. Finally, in step 6, the final C_{NOT} gate can be executed.

Noise of Quantum Computers: QCs are fickle devices. Currently, quantum computing is in the Noisy Intermediate-Scale Quantum (NISQ) era. NISQ-era QCs are prone to noise and variability which affect their reliability. While quantum error correcting codes (QEC) exist, NISQ-era QCs are too resource-constrained to implement QEC [12]. The main sources of error for NISQ-era QCs are from quantum decoherence and erroneous gate operations.

The first source of error is due to a phenomenon known as decoherence. In quantum computing, the collection of qubits used to carry out computations are the ideal quantum system. However, the system is not ideal since it also interacts with the environment (i.e. there is no way to isolate the qubits from the environment). Eventually, a system of qubits succumbs to the external noise of the surrounding environment and loses its quantum coherence.

Decoherence can be measured via T_1 and T_2 coherence times. T_1 is the decay constant associated with how long it takes a qubit in the excited state $|1\rangle$ to decay to the ground state $|0\rangle$. T_1 measures the loss of energy of a quantum system [15]. T_2 is the decay constant associated with how long it takes a qubit to lose its phase. Table 3 and Table 4 explain how to measure T_1 and T_2 coherence times, respectively.

Steps to Measure T_1 Coherence	
1	Initialize a qubit to the state $\|0\rangle$.
2	Apply an X gate to transform its state to $\|1\rangle$.
3	Wait for time t and measure the probability of being in the state $\|1\rangle$.

Table 3: Steps to measure T_1 coherence, reproduced from [15].

Steps to Measure T_2 Coherence	
1	Initialize a qubit to the state $\|0\rangle$.
2	Apply an H gate to transform its state to $\frac{\|0\rangle + \|1\rangle}{\sqrt{2}}$.
3	Wait for time t, then apply an H gate again, and measure the probability of being in the state $\|0\rangle$.

Table 4: Steps to measure T_2 coherence, reproduced from [15].

For T_2, After some time t, it is expected that the state goes to $\frac{|0\rangle \pm |1\rangle}{\sqrt{2}}$ (since something in the environment likely caused the qubit to go to state $|0\rangle$ or $|1\rangle$, and applying the last H gate causes the state to go to $\frac{|0\rangle \pm |1\rangle}{\sqrt{2}}$ [15]). Figure 14 illustrates the expected results for T_1 and T_2 times on a 1-qubit system.

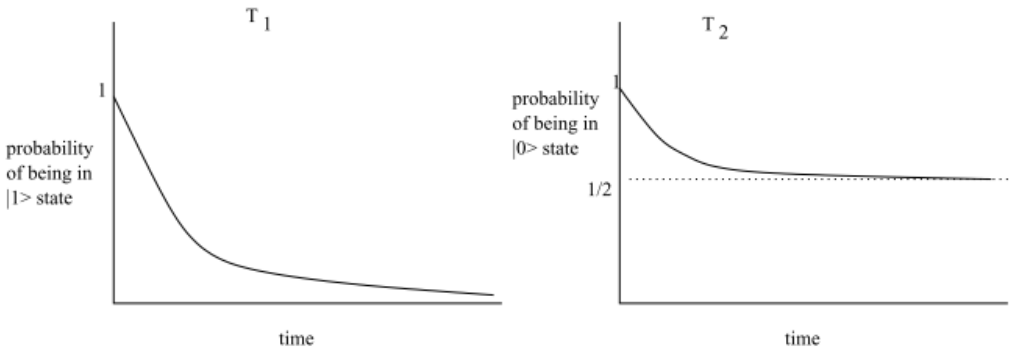


Figure 14: Expected results of experiments for T_1 and T_2. Reproduced from Chuang [15].

The second source of error is due to erroneous gate operations. When gates are applied to qubits in a QC, they do not always transform the qubit's state to the correct state-vector. For instance, an X gate is supposed to rotate a qubit's state by π radians about the x-axis. However, this operation might be erroneous, and instead rotate the qubit's state by $\pi + \epsilon$ radians. For IBM's QCs, single-qubit gates have associated error rates of order 10^{-3}, and two-qubit gates have associated error rates of order 10^{-2} [4]. Since two-qubit gate error rates are an order of magnitude larger than single-qubit gate error rates, two-qubit gates usually dominate the overall

error incurred by gate operations. Figure 15 shows associated error rates for single-qubit and two-qubit operations on the IBM QX2.

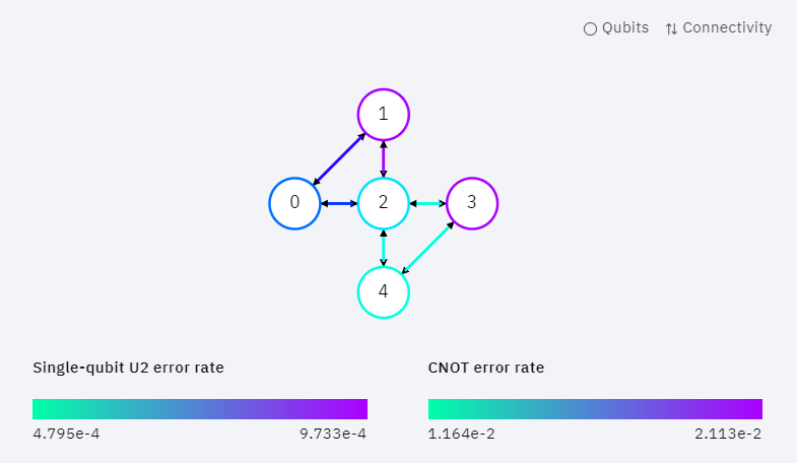


Figure 15: IBMQ QX2 topology and associated single-qubit and two-qubit error rates. Reproduced from "IBM Quantum Experience" [16].

Empirical Difference Between Naïve and Intelligent Transpilation: Consider the following quantum circuit to be executed on the IBM QX2 backend.

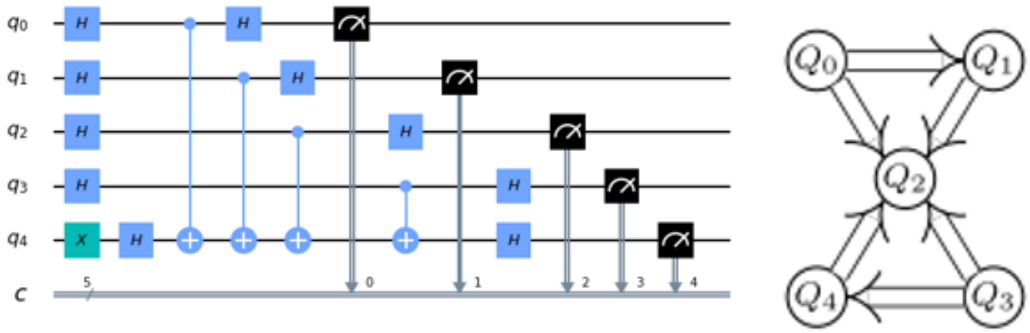


Figure 16: Circuit to transpile (left) on backend QC (right) from [11].

This circuit ideally has one correct state vector which is $|11111\rangle$. That is, after this circuit executes, each qubit should be in the excited state $|1\rangle$. Figure 17 and Figure 18 show two transpilations of the circuit to the backend.

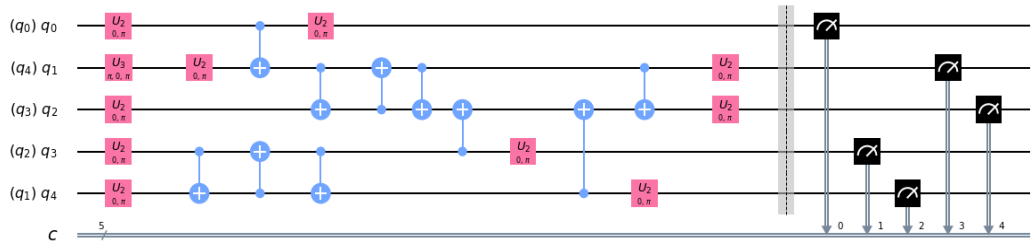


Figure 17: A naïve transpilation of the circuit in Figure 16 to the IBM QX2 backend.

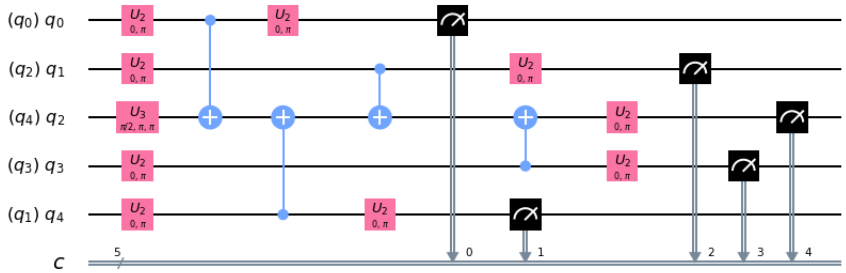


Figure 18: An intelligent transpilation of the circuit in Figure 16 to the IBM QX2 backend.

First note that both circuits are functionally equivalent (meaning in an ideal case, they produce the same state-vector). Next, notice that even though they are functionally equivalent, the first has significantly more gates than the second. As discussed earlier in this section, since there is a chance that each gate produces an erroneous result, requesting more gates increases the probability that an error will occur. Figure 19 shows the results of running the naïvely and intelligently transpiled circuits on the IBM QX2.

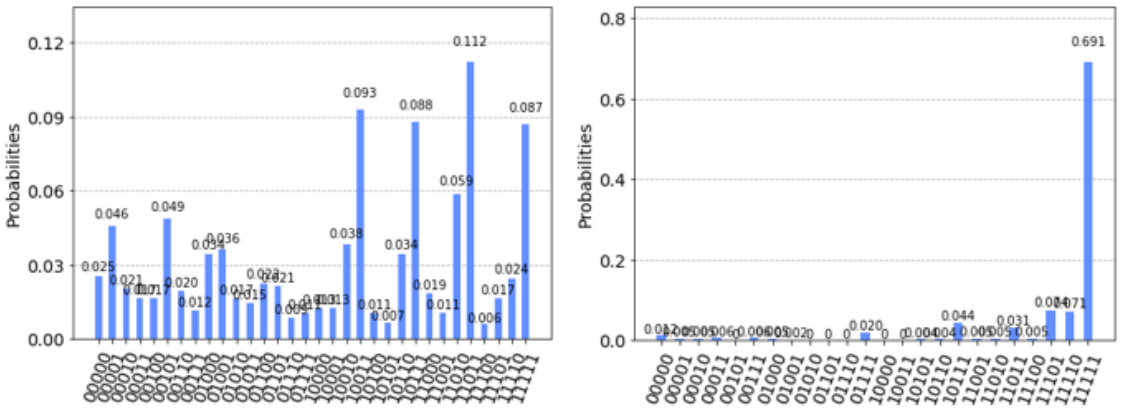


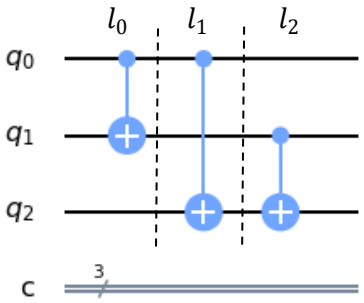
Figure 19: Results of running the naïvely transpiled circuit (left) versus intelligently transpiled circuit (right) on the IBM QX2.

Figure 19 shows that the naïvely transpiled circuit obtains the correct state-vector 8.7% of the time while the intelligently transpiled circuit obtains the correct state-vector 69.1% of the time. This example shows the necessity of intelligent QPT. Intelligent QPT maximizes the probability that a circuit produces the correct results when run on NISQ-era devices.

2.3.2 Quantum Layout Problem

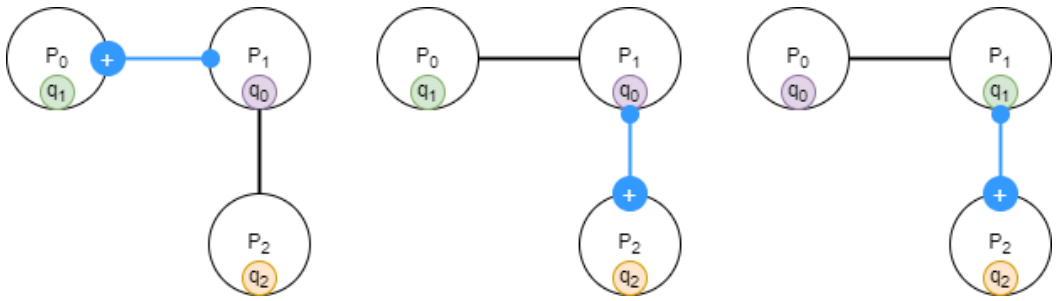
Before defining the QLP, a more methodical approach to addressing C_{NOT} constraints is explained. Steps 1-4 below provide a pseudo algorithm to address C_{NOT} constraints.

Step 1: Break the circuit into layers (this process is defined in Section 2.2.3).



Step 2: For each layer l_i, assign a physical-to-virtual (P2L) mapping λ_i such that all C_{NOT} constraints are satisfied in l_i. For N physical qubits and M logical qubits, let $V = \{P_0, P_1, \dots, P_{N-1}\}$ be the set of all physical qubits and $T = \{q_0, q_1, \dots, q_{M-1}\}$ be the set of all logical qubits. Now, λ is defined as $\lambda : V \to T$.

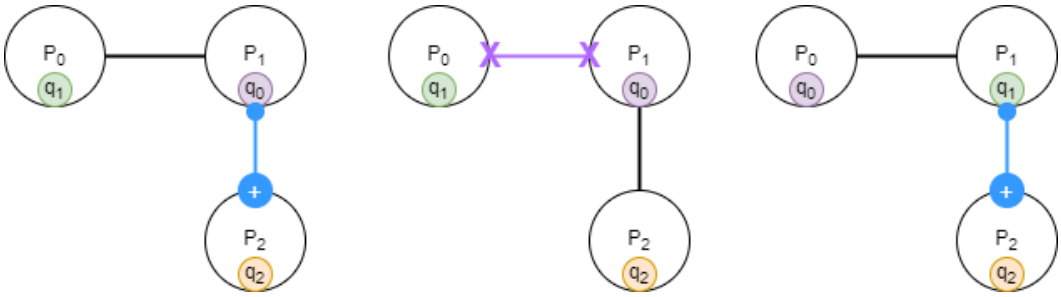
Satisfying layout λ_0 for l_0 | Satisfying layout λ_1 for l_1 | Satisfying layout λ_2 for l_2



These layouts are satisfying because all C_{NOT} operations occur between adjacent physical qubits. In this example, $\lambda_0 = \lambda_1 = \begin{pmatrix} P_0 & P_1 & P_2 \\ q_1 & q_0 & q_2 \end{pmatrix}$ and $\lambda_2 = \begin{pmatrix} P_0 & P_1 & P_2 \\ q_0 & q_1 & q_2 \end{pmatrix}$.

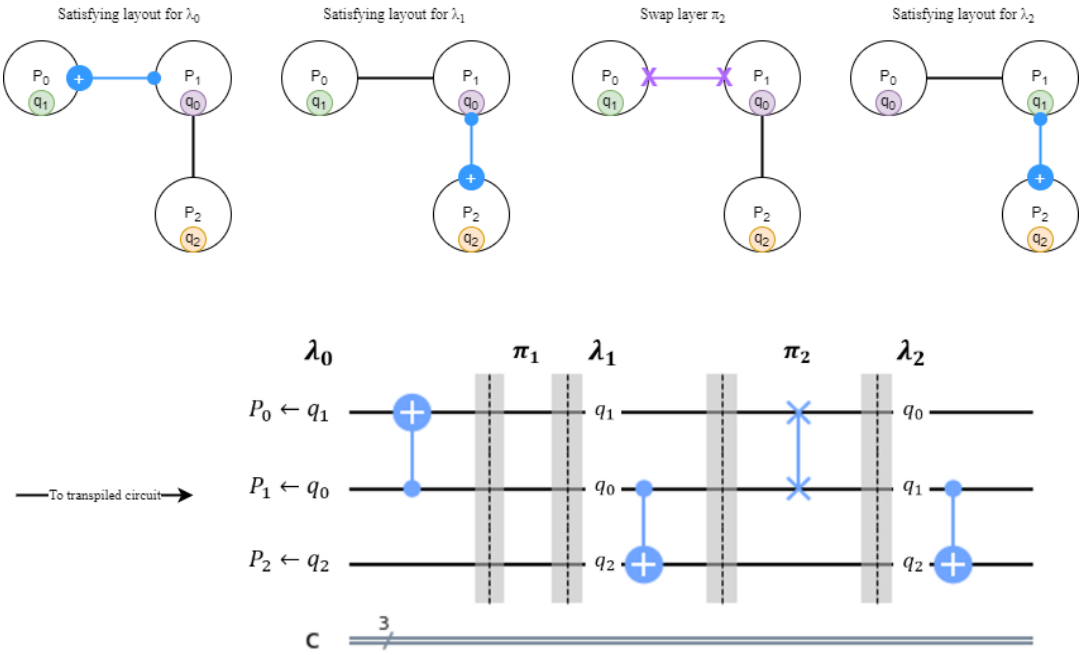
Step 3: Insert layers of SWAP gates π_i to permute between each λ_{i-1} and λ_i. In this example, since $\lambda_0 = \lambda_1$, no SWAP operations are required to permute between λ_0 and λ_1. However, $\lambda_1 \neq \lambda_2$. Thus, a sequence of SWAP operations must be found to transform $\lambda_1 \to \lambda_2$.

Satisfying layout λ_1 for l_1 | Swap layer π_2 to permute λ_1 to λ_2 | Satisfying layout λ_2 for l_2



As illustrated, to permute $\lambda_1 \rightarrow \lambda_2$ a SWAP must be performed between physical qubits P_0 and P_1. Note that a SWAP layer π_i may consists of no SWAP operations (e.g. π_1 in above example), or multiple $SWAP$ operations. As will be explored in later sections, finding a sequence of SWAPs to permute between two configurations is generally a difficult task.

Step 4: Convert satisfying layouts and SWAP layers into a transpiled circuit.



In the transpiled circuit, each horizontal line represents a physical qubit (as opposed to an abstract logical circuit where the horizontal lines represent logical qubits). In QPT, logical qubits are "loaded" into each physical qubit via the λs. Then, π layers permute the logical qubits along the physical qubits. Summarizing, first satisfying λs are chosen for each layer of the circuit. Then, sequences of SWAP operations are inserted to permute each λ_{i-1} to λ_i. The resulting transpiled circuit, after gate decomposition (i.e. the inserted SWAP gates must be converted into C_{NOT} operations), is able to execute on the specified backend QC. Moreover, the functionality of the

original circuit is preserved, meaning the transpiled circuit and original abstract circuit are functionally equivalent.

The Quantum Layout Problem: At this point, a solution to the QLP can be defined. Zulehner et al. define a solution to the QLP M_h as follows [11]:

$$M_h = (\lambda_0, \pi_1, \lambda_1, \pi_2, \dots, \lambda_{L-2}, \pi_{L-1}, \lambda_{L-1}) : \lambda_i \vdash \mathrm{l_i} \quad (2.9)$$

where L is the number of layers in the logical circuit.

Thus, to find solutions to the QLP, first satisfying layouts $\lambda_0 \dots \lambda_{L-1}$ (denoted by constraints $\lambda_i \vdash \mathrm{l_i}$ in M_h) must be found for each layer l_i of the circuit. Then, based on the λs, SWAP layers $\pi_1 \dots \pi_{L-1}$ must be found, where each π_i permutes λ_{i-1} to λ_i.

The QLP is the problem of mapping logical qubits of a quantum circuit to physical qubits of a backend QC such that the mappings satisfy all C_{NOT} constraints. Here, the mappings refer to the λs defined above. This is a loose definition of the QLP. Stricter definitions add one of the following requirements:

1. The satisfying λs result in SWAP layers of minimum aggregate cardinality. That is, $\sum_{\pi_i \in M_h} |\pi_i|$ is minimized.
2. The satisfying λs result in a transpiled circuit which executes with the maximum probability of success.

Unfortunately, to solve either approach optimally is NP-Hard, as proven by Siraichi et al. [13], and Tan and Cong [17]. The following section provides SOTA QLP-solvers that heuristically find solutions to the aforementioned definitions of the QLP.

2.3.3 State of the Art Optimization Techniques for the QLP

In defining the QLP, two stricter definitions were proposed in Section 2.3.2 that were both NP-Hard to optimally solve. The first, commonly known as finding *depth-optimal* solutions to the

QLP, were the first to be explored by researchers. The second, for which this research calls finding *fidelity-optimal* solutions to the QLP, were later explored. In this section, SOTA *depth-optimal* and *fidelity-optimal* QLP-solvers are discussed.

In 2018, Zulehner et al. proposed a novel *depth-optimal* QLP-solver [11]. Their approach was the first to use *layering* to finding solutions to the QLP. In a layered-approach, the input circuit is first divided into layers (as defined earlier in this section), then compliant mappings are found for each layer of the circuit. Their work is particularly ground-breaking as layering is a common step taken in subsequently devised QLP-solvers. They then define a solution to the QLP as a sequence $M_h = (\lambda_0, \pi_1, \dots, \lambda_{L-2}, \pi_{L-1}, \lambda_{L-1})$ where each λ_i is a satisfying P2L mapping for l_i (i.e. each λ_i satisfies all C_{NOT} constraints of l_i), and each π_j is a sequence of SWAP operations to permute between each λ_{j-1} and λ_j.

Next, Zulehner et al. define their objective as finding an M_h where each $\lambda_i \vdash \mathrm{l_i}$ and the distance between each λ_{j-1} and λ_j is minimized (thus, this is a *depth-optimal* approach). As the search space to find such an M_h is exponential, they use an A^* search algorithm to efficiently prune for high-quality satisfying solutions. In their approach, they start with the mapping of the previous layer λ_{i-1} and generate a minimum sequence of SWAP operations to permute to a compliant λ_i. Their approach generates locally optimal solutions (i.e. between a given λ_{i-1} and λ_i, $|\pi_i|$ is minimized), but does not guarantee globally optimal solutions (i.e. $\sum_{\pi_i \in M_h} |\pi_i|$ is not always minimized). To overcome this issue, they use a look-ahead scheme to account the estimated cost of π_{i+1} into the cost function of their A^* search algorithm. They further improve their look-ahead scheme by defining an initial placement scheme that yields a good guess for λ_0 (the initial mapping) by considering upcoming C_{NOT} gates early in the circuit.

In 2019, Li et al. proposed another *depth-optimal* QLP-solver which uses a SWAP-based Bidirectional (SABRE) heuristic search algorithm to find solutions to the QLP [18]. Li et al.

noticed that previous work, mainly from Zulehner et al. [11], suffered from high complexity, poor initial mapping, and limited flexibility. To address the shortcomings of previous efforts, Li et al. developed a SABRE-based QLP-solver. Their algorithm is provided in Figure 20 below.

Algorithm 1: SABRE's SWAP-based Heuristic Search

Input: Front Layer F, Mapping π, Distance Matrix D, Circuit DAG, Chip Coupling Graph $G(V, E)$
Output: Inserted SWAPs, Final Mapping π_f

```
while F is not empty do
    Execute_gate_list = ∅ ;
    for gate in F do
        if gate can be executed on device then
            Execute_gate_list.append(gate);
        end
    end
    if Execute_gate_list ≠ ∅ then
        for gate in Execute_gate_list do
            F.remove(gate);
            obtain successor gates from DAG;
            if successor gates' dependencies are resolved
             then
                F.append(gate);
            end
        end
        Continue;
    else
        score = [];
        SWAP_candidate_list =
         Obtain_SWAPs(F, G);
        for SWAP in SWAP_candidate_list do
            π_temp = π.update(SWAP);
            score[SWAP] =
             H(F, DAG, π_temp, D, SWAP);
        end
        Find the SWAP with minimal score;
        π = π.update(SWAP);
    end
end
```

Figure 20: SABRE algorithm. Reproduced from Li et al. [18].

Their algorithm differs from previous QLP-solvers in that first it is run several times to yield a high-quality initial mapping for λ_0. They do this by traversing the reverse circuit, where they not only consider gates at the beginning of the circuit to generate λ_0, but also gates in the rest of the circuit [18]. This differs from the QLP-solver devised by Zulehner et al. in which their initial mapping only considers gates at the beginning of the circuit. Next, Li et al.'s routing method improves upon that of Zulehner et al. by considering a candidate SWAP list, which they use to (in conjunction with a heuristic cost function) to make informed decisions on the best SWAP operations to perform in order to execute more gates. They notice that the routing method of

Zulehner et al. considers many candidate SWAPs that are redundant or unnecessary. Their informed routing method also operates with time complexity $O(n)$ as opposed to that of Zulehner et al. [11] which operates with time complexity $O(\exp(n))$ [18]. Their novel layout and routing methods, as compared to the best-known at the time by Zulehner et al., have an exponential speedup and comparable to better results on various benchmark QLP problem-instances [18].

In 2019, the first *fidelity-optimal* QLP-solvers were proposed by Tannu et al. [4] and Murali et al. [12]. In their approaches, rather than focusing on minimizing the number of SWAP gates inserted in the transpiled circuit, their objectives seek to maximize the probability that the transpiled circuit executes successfully. In both works, noise-data regarding the reliability of 1-qubit and 2-qubit operations is considered when determining mappings for each λ_i.

In Murali et al., their strategy to solve the QLP is to first find an initial mapping for λ_0 such that the mapping places logical qubits in physical qubits with high reliability. Next, they schedule gates in the circuit such that the circuit finishes execution within the coherence time of the qubits. Finally, they seek to minimize the number of SWAP gates inserted into the transpiled circuit [12]. In their research, they devise two QLP-solvers. In the first, they pose the QLP as a constrained optimization problem and use Satisfiability Modulo Theory (SMT) solvers to find solutions. In the second, they use greedy heuristics to find solutions to the QLP.

In Murali et al.'s SMT-based QLP-solver, they define variables to for P2L mappings (i.e. λ_is), gate start times, and routing paths used in the π layers. Next, they define constraints to ensure each λ_i is a valid P2L mapping (the mapping must be bijective), gates are executed in the correct order, and routing paths do not overlap [12]. Finally, they define an objective function which is to maximize the probability of successful circuit execution based on the calibration data from IBM.

Murali et al. admit that their SMT-based QLP-solver's time complexity scales poorly. Accordingly, they also devise two greedy heuristics-based QLP-solvers. In their greedy

heuristics-based QLP-solvers, both greedily choose the highest fidelity SWAP paths for the qubit routing policy. For initial placement, they devise a $GreedyV$ strategy which places logical qubits involved in more C_{NOT}s in strong subgraphs of the physical topology with high 2-qubit and readout fidelities. In their $GreedyE$ strategy, they place logical qubit pairs which frequently entangle in physically adjacent qubits that have high 2-qubit and readout fidelities. No direct comparison of their devised QLP-solvers are made against Li et al.'s [18], however their QLP-solvers outperform Qiskit's baseline in all test cases, with average performance gains of 2.9x and up to 18x over Qiskit [12].

2.4 Qiskit: A Quantum Computing Framework

Qiskit is an open-source framework that allows quantum researchers to create and execute (on either circuit on either a real QC or a simulator) quantum circuits. Moreover, Qiskit handles QPT, which in turn drastically increases the productivity of quantum researchers. Qiskit consists of four foundational elements: Terra, Aer, Ignis, and Aqua [10]. In this research, only Qiskit Terra and Aer are used. At a high level, Qiskit Terra provides quantum researchers with the ability to create quantum circuits, optimize them for a particular QC, and manage execution of batches of experiments [10]. Qiskit Aer provides QC simulators to aid quantum researchers. Sections 2.4.1 and 2.4.2 provide details on the core modules of Qiskit Terra and Qiskit Aer, respectively. Finally, Section 2.4.3 defines Qiskit's transpilers.

2.4.1 Qiskit Terra

Qiskit Terra is the "foundation on which the rest of Qiskit lies" [10]. Qiskit Terra's $qiskit.circuit$ module allows a user to construct quantum circuit objects. Sequences of gates can then be applied to a set of logical qubits in the user's quantum circuit object. Furthermore, measurement gates can be applied to read the states of logical qubits into classical bits. Algorithm 1 shows the construction of a Bell-state (bs) circuit in Qiskit.

Algorithm Construction of a Bell-State circuit in Qiskit.

Imports: Qiskit (qk) ;

/* Initialize circuit with 2 logical qubits and 2 classical bits */
$circuit = qk.QuantumCircuit(2,2)$;

$circuit.h(0)$; /* Apply Hadamard gate to 1st logical qubit */
$circuit.cx(0,1)$; /* Apply C_{NOT} gate where q_0 is the control and q_1 is the target */

/* Measure 1st and 2nd logical qubits into 1st and 2nd classical bits, respectively */
$circuit.measure([0,1],[0,1])$;

Output A $QuantumCircuit$ object which encapsulates a Bell-State circuit.

Algorithm 1: Bell-state circuit in Qiskit.

Next, Qiskit Terra's $qiskit.providers$ module allows a user to define a backend QC to execute their quantum circuit on. In this module, a $Provider$ is an object that provides access to various backend QCs [10]. A $Provider$ interacts with a backend QC and obtains information such as noise-data (e.g. 1-qubit, 2-qubit, and readout error rates), the coupling map of the QC, etc. A $Backend$ is an object that represents the QC (or simulator) [10]. In particular, a $Backend$ object has a $run()$ method which takes a transpiled circuit and executes it on the specific $Backend$ instance. Algorithm 2 shows the instantiation of a $Provider$ and $Backend$ in Qiskit.

Algorithm Instantiating a provider and backend in Qiskit.

Imports: Qiskit (qk) ;

/* $get\_provider()$ returns a $Provider$ object; user req'd to create account with IBMQ */
$provider = qk.IBMQ.get\_provider(hub = 'userHub', group = 'userGroup',$
$project = 'userProject')$;
/* $get\_backend()$ returns a $Backend$ object */
$backend = provider.get\_backend('ibmq\_5\_yorktown')$;

Output A $Backend$ object encapsulating a QC to run a circuit on.

Algorithm 2: Provider and backend configuration in Qiskit.

After a circuit is constructed and a backend is instantiated, the circuit is transpiled via methods and mechanisms available in Qiskit Terra's $qiskit.transpiler$ module. In Qiskit, QPT is handled

via a modular mechanism called a $PassManager$. A $PassManager$ is composed of a collection of transpilation passes which are applied to the circuit. For example, a $PassManager$ commonly has passes to address gate decomposition and C_{NOT} constraints (see Section 2.4.3 for more information about passes). Qiskit Terra also has a $transpile()$ method that takes a $QuantumCircuit$, $Backend$, and optimization level and automatically creates and executes a $PassManager$ on the circuit. Thus, the $transpile$ method handles all QPT steps and create a satisfying circuit that is executable on the selected backend. The optimization level dictates which passes are applied to the circuit (see Section 2.4.3 for more information). Algorithm 3 shows how to use the $transpile$ method to create a transpiled circuit, optimized for a given backend.

Algorithm Transpiling a circuit in Qiskit.

Imports: Qiskit (qk) ;
Inputs: A $QuantumCircuit$ to execute, a $Backend$ to execute on, and an opt_level.

$trans\_circ = qk.transpile(circuit, backend, optimization\_level = opt\_level)$;

Output A $QuantumCircuit$ object encapsulating a functionally equivalent transformation of the input circuit optimized to execute on the input backend.

Algorithm 3: Transpiling a circuit in Qiskit.

If opt_level =3, then all optimization passes available in Qiskit are executed, and this optimization level embodies the SOTA transpiler offered by Qiskit. Finally, the transpiled circuit can be executed on the backend via the backend's $run()$ method. As execution is not synchronous, this method returns a Job object. The job, once complete, returns a $Result$ object, which contains the results of running the circuit on the backend QC. Algorithm 4 shows how to submit a job and extract the results of running the circuit on the backend.

Algorithm Running a transpiled circuit on a backend in Qiskit.

Imports: Qiskit (qk) ;
Inputs: A $QuantumCircuit$ encapsulating the transpiled circuit to execute, and a $Backend$ to execute on.

/* Before submitting a job, serialize circuit and backend into a $QObj$ */
$q\_obj = qk.assemble(trans\_circ, backend)$;

/* Submit the job and receive Job object */
$job = backend.run(q\_obj)$;

/* Monitor for completion of job */
$qk.tools.monitor.job\_monitor(job)$;
/* Above command returns when job complete */

/* Retrieve results of job */
$results = job.results().get\_counts(trans\_circ)$;

Output A dictionary where the keys are measured basis states and the values are the number of shots that returned each basis state.

Algorithm 4: Transpiling and running a circuit in Qiskit.

In Algorithm 4, first the transpiled circuit and backend are serialized into a $QObject$ (this is the format that the $run()$ method expects). Then, the job is submitted via Qiskit's $run()$ method. Next, a job monitor is used to wait for the backend to return a $Result$ object. Finally, a $Result$ object is obtained. By default, a Qiskit program executes $shots = 1024$ times on the backend QC. As such, the results obtained from Algorithm 4 are encapsulated in a dictionary where the keys are the measured basis states, and the values are the number of shots which returned each basis state.

2.4.2 Qiskit Aer

Qiskit Aer's primary functionality is the speed up development time by providing quantum researchers with QC simulators. While real quantum hardware provide the most authentic results, simulators are demanded for two main reasons:

1. Reproducibility of experiments

2. Long wait times in queue

First, when quantum researchers perform experiments, results inevitably vary due to the stochastic noise of QCs. Effectively, it is impossible to accurately reproduce results unless the state of the QC at the time of program execution is determinable (which is currently unfeasible). Second, and most frustratingly, for many of IBM's backend QCs, the wait time to actually run an experiment is exceptionally long (from personal experience, some queue times are over 24 hours).

To address these demands, Qiskit Aer provides a $QasmSimulator$ [10]. A base $QasmSimulator$ can be thought of as a backend QC with connectivity defined by κ_N (i.e. for N physical qubits, all physical qubits are connected to each other) and perfect fidelity (i.e. all gate operations produce the correct state-vectors without noise and environmental noise, such as T_1 and T_2 errors). Algorithm 5 shows how to run the Bell-State circuit on the $QasmSimulator$.

Algorithm Running Bell-State circuit on simulator in Qiskit.

Imports: Qiskit (qk) ;

/* Create Bell-State circuit as done previously in this section */
$circuit = create\_bell\_state\_circuit()$;

/* Get the simulator backend */
$sim = qk.Aer.get\_backend('qasm\_simulator')$;

/* Submit the job; execute automatically performs $transpile()$ and $assemble()$ */
$job = qk.execute(circuit, sim)$;

/* Retrieve results of job */
$results = job.results().get\_counts(circuit)$;

Output A dictionary where the keys are measured basis states and the values are the number of shots that returned each given basis state.

Algorithm 5: Running a circuit on Qiskit's backend simulator.

In some experiments, the noisiness of QCs is a variable of the research effort. However, the variability of real QCs make results irreproducible. Qiskit Aer provides a means to address the

communities demand for consistently unreliable QC simulators via a $NoiseModel$. A $NoiseModel$ can augment a $QasmSimulator$ and provides it with configurable noise data [10]. A quantum researcher can customize their own $NoiseModel$, or derive one from a specific backend QC. While a $NoiseModel$ does not perfectly mimic a given backend QC, it performs similar by accounting for gate-related errors (1-qubit, 2-qubit, and measurement), as well as T_1 and T_2 related errors [10]. Algorithm 6 shows how to augment a $QasmSimulator$ with a $NoiseModel$ for a given backend QC.

Algorithm Running Bell-State circuit on simulator with a NoiseModel in Qiskit.

Imports: Qiskit (qk), qk.aer.NoiseModel (NoiseModel) ;
Inputs: A $Backend$ object encapsulating the QC to mimic.

```
/* Create Bell-State circuit as done previously in this section */
circuit = create_bell_state_circuit() ;

/* Create a noise model for the backend */
noise_model = NoiseModel.from_backend(backend) ;
noise_model_dict = noise_model.to_dict() ;    /* Can save noise model for later use */

/* Get the simulator backend */
sim = qk.Aer.get_backend('qasm_simulator') ;

/* Get coupling map and basis gates for backend */
coupling_map = backend.configuration().coupling_map ;
basis_gates = noise_model.basis_gates ;

/* Submit job */
job = qk.execute(circuit, sim, noise_model = noise_model,
                 coupling_map = coupling_map, basis_gates = basis_gates,
                 optimization_level = 3, seed_transpiler = 1, seed_simulator = 1) ;

/* Retrieve results of job */
results = job.results().get_counts(circuit) ;
```

Output A dictionary where the keys are measured basis states and the values are the number of shots that returned each given basis state.

Algorithm 6: Augmenting Qiskit's backend simulator with a NoiseModel.

In Algorithm 6, first notice that the noise model can be saved for later use. That is, $noise\_model\_dict$ can be used to instantiate the same $NoiseModel$. Next, since the Qiskit

transpiler and simulator are stochastic, the $execute()$ command must be provided seeds if consistent results are required. The output of running a circuit on a simulated noisy backend mimics executing the circuit on the real backend, accounting for decoherence, single and two-qubit gate errors, and measurement errors.

2.4.3 Qiskit's Transpiler

Qiskit's transpiler is implemented via a $PassManager$. A $PassManager$ iterates through a series of passes, where each pass either creates a transformation of the circuit, or runs analysis on the circuit. A minimal $PassManager$ first runs a decomposition pass (to convert all gates to the devices basis gate-set), then a pass to ensure all C_{NOT} constraints are satisfied, and finally a final decomposition pass (to convert all inserted gates from the previous pass to the devices basis gate-set). Qiskit offers four pre-defined $PassManagers$, optimization levels 0 through 3, where the optimization level 0 $PassManager$ does minimal optimization, and optimization level 3 $PassManager$ applies a series of SOTA passes to optimize a circuit for a given backend QC [10]. Figure 21 provides the passes Qiskit applies in its baseline $PassManager$ (optimization level 1) and its optimal $PassManager$ (optimization level 3). Let QST_i denote the Qiskit transpiler with optimization level i.

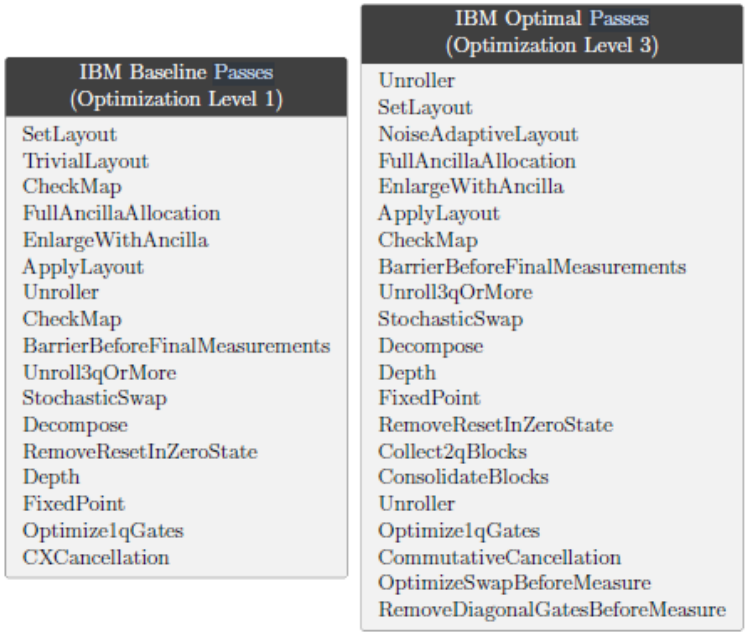


Figure 21: Transpiler passes executed by Qiskit optimization levels 1 and 3. Reproduced from Kamaka [19].

Qiskit's SOTA transpiler (QST_3) is composed of a series of transpiler passes which collectively exhibit the SOTA transpilation tool available to Qiskit users. The transpiler passes can be divided into three main categories: decomposition passes, gate-consolidation passes, and QLP-solver passes. The gate decomposition passes decompose the gates of the abstract circuit into gates which are realizable on quantum hardware. The gate-consolidation passes combine sequences of operations into new sequences of operations which are shorter, mainly using circuit identities (e.g. $HXH = Z$, $HYH = -Y$, and $HZH = X$ [7]).

In Qiskit, the QLP is divided into two transpiler passes: first, a satisfying initial mapping of between logical and physical qubits (i.e. a solution for λ_0) is established via a $layout_{method}$ pass. Second, all subsequent mappings (i.e. $\lambda_1, \dots, \lambda_{L-1}$) are established, as well as SWAP gates inserted, via a $routing_{method}$ pass. In the QST_3, the $layout_{method}$ and $routing_{method}$ passes, by default, are the $DenseLayout$ and $StochasticSwap$ passes, respectfully.

While the QST_3 has a series of pre-defined QPT steps, it can easily be augmented to execute additional/alternative passes if needed. For instance, the $DenseLayout$ pass can be replaced by a $NoiseAdaptiveLayout$ pass which uses a different algorithm for the $layout_{method}$ than the former. The modularity of the $PassManager$ allows quantum researchers to easily test various transpiler configurations, which is particularly useful in this research effort when comparing the effectiveness of the devised QLP-solvers versus SOTA QLP-solvers. Table 5 and Table 6 provide the $layout_{method}$ and $routing_{method}$ passes available in Qiskit.

Layout Method	Description
$TrivialLayout$	Each physical qubit is mapped to each logical qubit in increasing order [10]. For instance, for 3 logical and physical qubits, $\{P_0 \rightarrow q_0, P_1 \rightarrow q_1, P_2 \rightarrow q_2\}$.
$DenseLayout$	Logical qubits are mapped to physical qubits by placing the most frequently entangled logical qubits in the most connected subset of the physical qubits [10]. The cost function uses a *depth-optimal* strategy.
$NoiseAdaptiveLayout$	Employs the qubit mapping method of Murali et al. [12]. Their cost function uses a *fidelity-optimal* strategy.
$SabreLayout$	Employs the qubit mapping method of Li et al. [18]. Their cost function uses a *depth-optimal* strategy.

Table 5: Layout methods available in Qiskit.

Routing Method	Description
$Basic$	Naively routes logical qubits along physical qubits to satisfy C_{NOT} constraints (in a greedy manner).
$Lookahead$	Employs the qubit routing method of Zulehner et al. [11]. Their cost function uses a *depth-optimal* strategy.
$Stochastic$	Using a randomized algorithm, iteratively tests random permutations of P2L mappings and selects lowest-cost permutation at each step [19]. The cost function uses a *depth-optimal* strategy.
$Sabre$	Employs the qubit routing method of Li et al. Their cost function uses a *depth-optimal* strategy.

Table 6: Routing methods available in Qiskit.

2.5 Metaheuristics

Talbi defines metaheuristics as a family of approximation techniques used to find high-quality solutions to difficult optimization problems [6]. Boyd and Vandenberghe define an optimization problem as follows [20]:

$$\begin{aligned} &minimize\ f(x) \\ &subject\ to \\ &g_i(x) \leq 0, \qquad i = 1, \dots, m \\ &h_j(x) = 0, \qquad j = 1, \dots, p \end{aligned}$$

Here, x is an n-variable vector of real and/or discrete values, f is the objective function (also known as a fitness function) to optimize where $f : (\mathbb{Z} \vee \mathbb{R})^n \rightarrow \mathbb{R}$, $g_i(x) \leq 0$ are the inequality constraints, $h_j(x) = 0$ are the equality constraints, and $m \geq 0, p \geq 0$ [20]. Each element (variable) of x is known as a *decision variable*. Although the above formulation minimizes the objective function, maximization is accomplished by negating the objective function. Thus, $maximize\ f(x)$ is the same as $minimize\ -1 \cdot f(x)$.

An optimization problem can be converted into a problem that metaheuristics can be applied to by defining a representation to encode solutions of the problem. In other words, a representation is a function $R : X \rightarrow S$, where S is the search space. Thus, R induces a search space on the solution space which metaheuristics can traverse. In literature, the search space is commonly called the *genotypic space* and the solution space is called the *phenotypic space* (and encoded solutions are *genotypes* while decoded solutions are *phenotypes*). According to Talbi, a representation R is sufficient if it meets the following criteria [6]:

1. Completeness: $\forall x \in X$, x is represented by R.
2. Connexity: $\forall s_1, s_2 \in S \times S$, there is a search path from s_1 to s_2 in S.
    a. This ensures that given any solution $s \in S$, a global optimum s^* is reachable.

3. Efficiency: The representation is easy to manipulate via search operators.

Once a sufficient representation function R has been defined, $R^{-1} : S \rightarrow X$ can be used to decode an encoded solution. Then, the feasibility of a solution is determined via the constraints, and the quality of a solution evaluated via the objective function.

Random-Key Encoding: To provide an example of encoding, consider the traveling salesman problem (TSP). For a TSP instance with n cities $c_1, c_2, \ldots, c_n$ (where c_1 is the start city), a feasible solution (i.e. a tour) is represented by a permutation of the list $t_k = (c_2, c_3 \ldots, c_n)$. Thus, the solution space X_{TSP} is the set of all permutations of t_k. How can all solutions in X_{TSP} be represented for a metaheuristic? Naively, $n-1$ decision variables $d_2, d_3, \ldots, d_n$ can be defined, where each d_i is assigned to a city c_j. However, such a representation is inefficient, as it produces many infeasible solutions (e.g. a city is visited more than once). Alternatively, a *random-key* encoding can be used. In *random-key* encoding, real-valued variables are used to represent permutations [21]. For the TSP, first define $n-1$ decision variables $d_2, d_3, \ldots, d_n$ where each d_i corresponds to c_i. Then, assign each d_i a real-value in range [0,1]. Finally, to decode a valid permutation from the decision variables, sort the elements of t_k by their corresponding d_is in ascending order (via a stable sorting algorithm). The resultant phenotype is always a valid permutation of t_k. The search space, S_{TSP}, is defined by the continuous space $\mathbb{R}^{n-1}$. Figure 22 shows how random-key encoding works on a sample TSP problem.

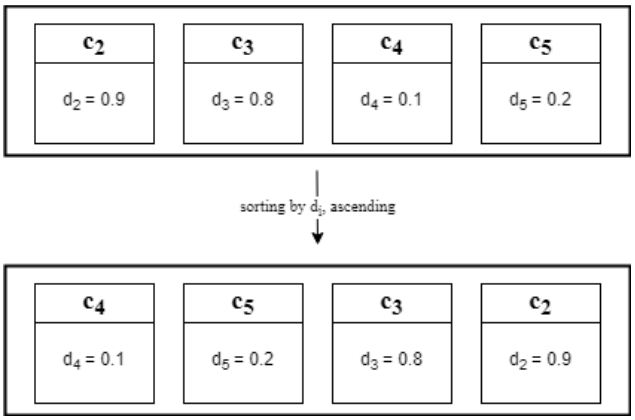


Figure 22: Example of how random-key encoding induces a permutation of a tour in a 5-city TSP problem-instance.

Metaheuristic algorithms are generally classified into two categories: single-solution-based and population-based metaheuristics. These classes of metaheuristics significantly differ in strategy employed to optimize a given problem. Sections 2.5.1 and 2.5.2 explain the single-solution based metaheuristics and population-based metaheuristics used in this research effort, respectively.

2.5.1 Single-Solution Based Metaheuristics

Single-solution based-metaheuristics (S-metaheuristics) iteratively improve a single solution. Talbi describes that this class of metaheuristics can be thought of as taking "walks" through the search space of the optimization problem [6]. During these walks, the goal is to move from the initial solution to a solution of better fitness.

In the design of S-metaheuristics, walking through the search space is commonly done via exploration of neighborhoods. A neighborhood function N is a function that assigns each solution $s \in S$ a set of solutions $N(s) \subset S$ [6]. Talbi says a neighbor $n \in N(s)$ is "generated by the application of a *move operator* that performs a small perturbation to the solution s." The perturbation a move operator performs is also known simply as a *move*. The structure of a neighborhood is dependent on the optimization problem. In order for S-metaheuristics to perform

meaningful search, the neighborhood should have strong locality. Locality is defined as the effect on the fitness when small perturbations are made in the representation [6]. Strong locality loosely means that $\forall(s \in S, s' \in N(s))$, $f(s)$ is similar to $f(s')$.

One of the most common S-metaheuristics used in practice is local search. In local search, one starts with an initial solution s_0. Then, a single neighborhood function N is defined. At each iteration, the algorithm replaces the current solution with a neighboring solution of better objective score than the current (until none exists) [6]. Algorithm 7 presents the local search algorithm.

Algorithm Template of a local search algorithm.

Inputs: A neighborhood function $N(s)$.

$s = s_0$; /* Generate an initial solution s_0 */
While not $Termination.Criterion$ **Do**
  $Generate\ N(s)$; /* Generation of candidate neighbors */
  If there is no better neighbor **Then** Stop ;
  $s = s'$; /* Otherwise, select a better neighbor $s' \in N(s)$ */
End While

Output Final solution (local optima).

Algorithm 7: Local search. Reproduced from Talbi [6].

Local search continues until some $Termination.Criterion$ are satisfied. In general, search ceases when for a given solution s, $\forall s' \in N(s)$, $f(s') \leq f(s)$ (for maximization problems) or $f(s') \geq f(s)$ (for minimization problems).

While local search is a useful S-metaheuristic, it tends to get stuck in local optima. To overcome this, Talbi proposes numerous strategies [6]. For this research effort, the technique of variable neighborhood descent (VND) is selected as a means to perform local search and escape local optima.

2.5.1.1 Variable Neighborhood Descent

VND improves upon local search by systematically changing the landscape of the optimization problem. Specifically, VND uses numerous neighborhood functions (as opposed to local search which uses only one neighborhood function). VND, like local search, is a deterministic algorithm. In VND, first a set of neighborhood structures $N_l(l = 1, \dots, l_{max})$ are defined. Next, an objective function to optimize is defined. Then, an initial solution s_0 is provided (or generated randomly). VND then searches N_1 for an improving neighbor until no improving neighbors are found in N_1. At which point, the neighborhood structure is changed from N_l to N_{l+1}. If an improving solution is found in N_l, the neighborhood structure returns to the first neighborhood (N_1). This process continues until no improving solutions are found in $N_1 \dots N_{l_{max}}$. Algorithm 8 presents the VND.

Algorithm Template of the VND algorithm.

Inputs: A set of neighborhood structures N_l for $l = 1, \dots, l_{max}$ and an objective function f to optimize.

$s = s_0$; /* Generate an initial solution s_0 */
$l = 1$;
While $l \leq l_{max}$ **Do**
  Find the best neighbor s' of s in $N_l(s)$;
  If $f(s') \leq f(s)$ **Then** $s = s'$; $l = 1$; /* for minimization problem */
  Otherwise $l = l + 1$;

Output Final solution found (local optima with respect to all neighborhoods).

Algorithm 8: VND. Reproduced from Talbi [6].

Mladenović and Pérez explain that VND is built upon the following perceptions [22]:

1. A locally optimal solution with respect to one neighborhood structure is not necessarily a locally optimal solution in a different neighborhood structure.
2. A globally optimal solution is locally optimal with respect to all neighborhood structures.

3. In many problems, locally optimal solutions with respect to one or several neighborhood structures are relatively close to each other.

In the development of metaheuristics to solve optimization problems, one must account for two competing objectives: intensification and diversification. Intensification refers to exploitation of the best solution(s) found. Diversification refers to exploration of the search space. Local search primarily focuses on intensification. Given an initial solution, local search iteratively selects the most improving neighbor until none exists. However, if the initial solution was in a region of the search space not close to a global optima, local search can reach a local optima that is not near (in fitness) to the global optimal. VND, like local search, intensifies a given solution until no improving solutions are found in N_1. Then, VND changes the neighborhood to search for improving solutions in different neighborhoods. By doing so, this allows VND to potentially escape the basin of attraction of local optima. In VND, diversification is managed via systematic neighborhood changes.

2.5.2 Population-Based Metaheuristics

While S-metaheuristics improve a single solution, population-based metaheuristics (P-metaheuristics) iteratively improve a population of solutions. Algorithm 9 presents a template for P-metaheuristics.

Algorithm High-level template of P-Metaheuristics.

$P = P_0$; /* Generation of initial population */
$t = 0$;
Repeat
  $Generate(P_t')$; /* Generation of a new population */
  $P_{t+1} = Select\_Population(P_t \cup P_t')$; /* Select new population */
  $t = t + 1$;
Until Stopping criteria satisfied

Output Best solution(s) found.

Algorithm 9: P-metaheuristics template. Reproduced from Talbi [6].

In P-metaheuristics, first an initial population of solutions is generated. Then, P-metaheuristics iteratively generate new populations of solutions and apply a selection operation to select the next generation's population from the current population and the newly spawned population.

Whereas S-metaheuristics focus more on intensification of a given solution, P-metaheuristics generally focus more on diversification by using an evolving population of candidate solutions. In this research effort, two P-metaheuristic approaches are explored: evolutionary algorithms and evolution strategies. Both approaches are stochastic bio-inspired techniques for optimization.

Evolutionary algorithms (EA): In EAs, first an initial population of solutions is generated. Generally, the initial population is generated by uniformly sampling the search space. Doing so allows for appropriate diversification. Next, each member of the population is scored via the fitness (objective) function. EAs employ the following operators each iteration (in the following order) [6]:

1. Selection: This operation determines which parents of the current population are chosen for the next generation with a bias towards better fitness.
2. Reproduction: In EAs, reproduction consists of the following two operations: mutation and crossover.
    a. Mutation: The mutation operator creates perturbations to members of the population to create new offspring. Mutation is necessary in order to manage diversity in the search process.
    b. Crossover: The crossover operator combines two or more solutions in the population to generate new offspring. Crossover is necessary in order to manage intensification in the search process. Crossover takes parents from the population,

mates them, and attempts to generate offspring which exhibit the best traits of their parents.

3. Replacement: This operation determines which members of the previous generation's population and newly spawned offspring from (2) move on to the next generation. Thus, replacement employs survival of the fittest in Darwinian evolution.

Algorithm 10 presents the high-level template for an EA.

Algorithm Template of an EA.

Generate($P(0)$) ; / Initial population */
$t = 0$;
While not $Termination.Criterion(P(t))$ **Do**
  $Evaluate(P(t))$;
  $P'(t) = Selection(P(t))$;
  $P'(t) = Reproduction(P'(t))$; $Evaluate(P'(t))$;
  $P(t+1) = Replace(P(t), P'(t))$;
  $t = t + 1$;
End While

Output Best individual or best population found.

Algorithm 10: Template for an EA. Reproduced from Talbi [6].

Genetic Algorithms (GA) and Evolution Strategies (ES): GAs and ESs are both types of EAs. Both metaheuristics are bio-inspired population-based stochastic optimization techniques. Table 7 shows the main differences between GAs and ESs.

Algorithm	Genetic Algorithms	Evolution Strategies
Developers	J. Holland	I. Rechenberg, H.-P. Schwefel
Original applications	Discrete optimization	Continuous optimization
Attribute features	Not too fast	Continuous optimization
Special features	Crossover, many variants	Fast, much theory
Representation	Binary strings	Real-valued vectors
Recombination	n-point or uniform	Discrete or intermediary
Mutation	Bit flipping with fixed probability	Gaussian perturbation
Selection (parent selection)	Fitness proportional	Uniform random
Replacement (survivor selection)	All children replace parents	(λ, μ) $(\lambda + \mu)$
Specialty	Emphasis on crossover	Self-adaptation of mutation step size

Table 7: Main characteristics of GAs versus ESs. For ES, λ is the number of offspring and μ is the number of parents. Reproduced from Talbi [6].

The first main difference between GAs and ESs is that the former are generally used for discrete optimization, while the latter are generally used for continuous optimization. Second, GAs traditionally use n-point or uniform crossover, while ESs typically use either no crossover or uniform crossover [6]. Third, GAs traditionally use a "bit-flipping" mutation while ESs use normally (Gaussian) distributed mutation [6] [23]. Fourth, replacement strategies are similar for both; either selecting members of the next generation based on the best (i.e. highest fitness) members of the children (non-elitist GA, (λ, μ)-ES) or children∪parents (elitist GA, $(\lambda + \mu)$-ES). Finally, in terms of overall structure, GAs evolve a population of solutions, where each member of the population is composed of tunable parameters. ESs also do this, except it is also popular to jointly evolve strategy parameters, where the strategy parameters manage mutation step size for each traditional parameter. As such, ESs are coined as employing *self-adaptation* [6].

2.5.2.1 Biased Random-Key Genetic Algorithm

A biased random-key genetic algorithm (BRKGA) is a variant of a canonical GA tailored to optimize problems whose solutions are encoded via random-keys. Before a BRKGA can execute, the following hyperparameters must be configured [24]:

1. Population size (p): The total number of individuals to make up the population for a given generation.
2. Elite population portion (p_e): Automatically persist $\lfloor p * p_e \rfloor$ most fit individuals of the current population to the next generation.
3. Mutant population size (p_m): Introduce $\lfloor p * p_m \rfloor$ random solutions into the next generation.
4. Bias (ρ_a): In crossover, children inherit genes from elite parent with probability $\rho_a > 0.5$ and from non-elite parent $1 - \rho_a$.

In addition, the BRKGA must be provided with an objective function(s) to optimize and the number of decision variables. Algorithm 11 presents the BRKGA.

Algorithm Template of a BRKGA.

Inputs: Population size p, elite population portion p_e, mutant population portion p_m, number of decision variables n, probability to inherit genes from an elite parent ρ_a, and an objective function to optimize f_{obj}.

/* Calculate size of elite population e and size of mutant population m */
$e = \lfloor p * p_e \rfloor$; $m = \lfloor p * p_m \rfloor$;

$f^* = \infty$; /* Initial value of best solution found */
$Generate(P)$; / Initial population P */

While not $Termination.Criterion$ **Do**
  $Evaluate(P)$; /* Evaluate current population via f_{obj} */

  /* Split population into elite and non-elite partitions via f_{obj} */
  $Partition\ P$ into two sets P_e and $P_{\bar{e}}$;

  $P^+ = P_e$; /* Initialize population of next generation */

  $Generate$ set P_m of mutants, each mutant with n random keys ;
  $P^+ = P^+ \cup P_m$; /* Add mutant population to next generation */

  $P_c = \{\}$; /* Initialize offspring population */
  For ($i = 0$; $i < n$; i++) **Do**
    $Select$ parent a at random from P_e ;
    $Select$ parent b at random from $P_{\bar{e}}$;
    For ($j = 0$; $j < n$; j++) **Do**
      Throw a biased coin with probability ρ_a of resulting heads;
      If $heads$ **Then** $c[j] = a[j]$;
      Otherwise $c[j] = b[j]$;
    End For
    $P_c = P_c \cup \{c\}$; / Add offspring c to offspring population */
  End For
  $P^+ = P^+ \cup P_c$; */ Add offspring population to next generation */
  $P = P^+$; /* Update population */

  Find best solution χ^+ in P ;
  If $f_{obj}(\chi^+) < f^*$ **Then** $f^* = f_{obj}(\chi^+)$;

End While

Output Best solution found f^*.

Algorithm 11: BRKGA. Adapted from Goncalves and Resende [24].

After the algorithm's parameters are configured, the BRKGA begins by partitioning the population P into two partitions, P_e and $P_{\bar{e}}$. The elite population, P_e, consists of the $e = \lfloor p * p_e \rfloor$

most fit members of P. The remaining members of P are placed in the non-elite population, $P_{\bar{e}}$. P_e is automatically added to the next generations population. Next, $m = \lfloor p * p_m \rfloor$ random solutions (mutants) are introduced into the next generations population. In BRKGAs, there is no direct mutation operator. Instead, mutation is mimicked by introducing random solutions into each population. Next, $p - e - m$ children are generated via crossover. In BRKGAs, a parameterized uniform crossover operator is used. In this crossover, a parent a is selected from P_e and a parent b is selected from $P_{\bar{e}}$. Then, traditional uniform crossover is applied between a and b, except the children inherit genes from a with probability ρ_a and genes from b with probability $1 - \rho_a$. Thus, there is a bias to select genes from the elite (more fit) parent. Finally, the population is replaced by $P_e \cup P_m \cup P_c$, where P_m is the mutant population and P_c are the offspring generated via crossover. The algorithm iterates until some $Termination.Criterion$ are satisfied.

2.5.2.2 Multi-Objective Evolution Strategies

Multi-Objective Evolution Strategies (MOES) is a software package which is capable of performing single-objective or multi-objective optimization of real and/or integer parameters. Moreover, MOES is efficient, written in C/C++ and is massively parallel (since evaluation of individuals in a population is an embarrassingly parallel task) [23]. Parallelism is achieved either through using MPI with compiled functions to evaluate the objectives, or in the so-called "offline" mode where a serial implementation of MOES communicates with an external parallel program through a series of Python scripts that read and write files. Furthermore, when performing single objective optimization, MOES can use a hybrid optimization algorithm, employing a local search (the Conjugate Gradient algorithm if a gradient is available, or otherwise the Nelder-Mead algorithm) after the ES portion of the calculation terminates [23]. Figure 23 shows the high-level master-slave architecture of MOES. The same architecture is employed when performing single objective optimization in MPI-mode, only the step to perform DEA analysis is omitted.

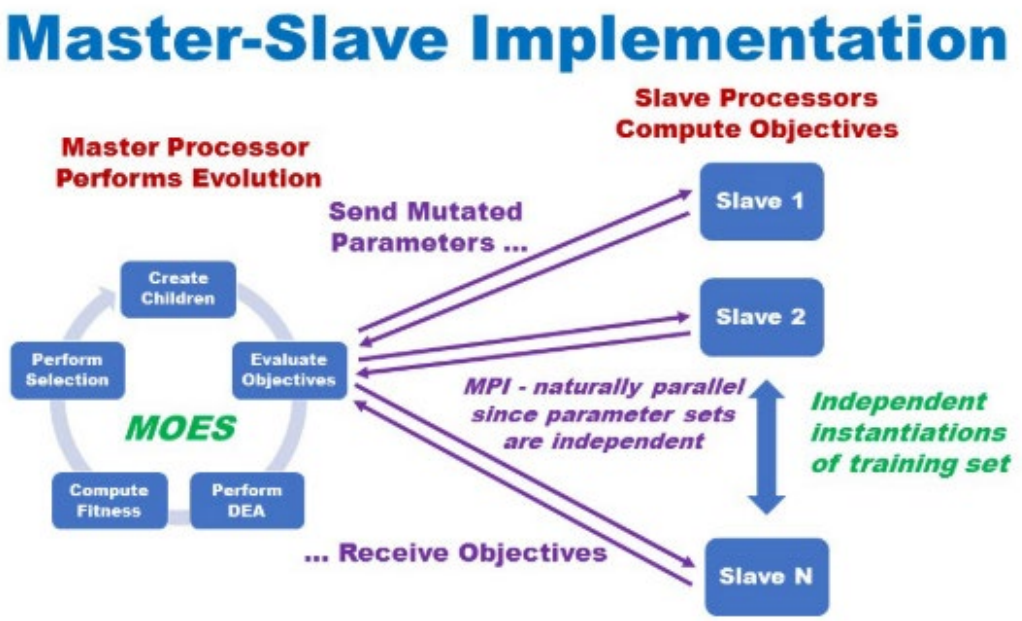


Figure 23: Master-slave architecture of MOES. Reproduced from Lill and Smith [23].

As is shown in Figure 23, the master processor handles all evolutionary operations, while the slave processors are used to evaluate the objective function on each member of the population. According to Lill and Smith, MOES requires the user to do the following in order to optimize a given problem:

1. Define objective function(s) in C/C++ function.
2. On the initial call, provide MOES with the number of objective functions and parameters, and whether each parameter is real or integer.
3. On the initial call, tell MOES whether to minimize or maximize each objective.
4. On the initial call, specify strict $[a, b]$ bounds and initial the standard deviations for each real parameter; initialize the mutation probabilities for each integer parameter.
5. On all subsequent calls, accept a set of mutated parameters from MOES and return the objective(s) to MOES.

On the master processor in Figure 23, the evolutionary process employed by MOES is shown. First, children are created via mutation and possibly crossover operations. Next, each member of the population is scored via the objective function. For multiple objective problems, Pareto-based fitness scores are computed using Data Envelopment Analysis [23]. Since this research effort seeks to optimize a single-objective optimization problem, DEA is not performed. Finally, selection is performed (i.e. which members of the population survive to the next generation) by simple truncation. This is standard in ES, but MOES also includes an option to perform tournament selection. This cycle continues until some termination criteria are satisfied.

MOES has numerous options to create new children. First, the parents can be simply mutated (asexual reproduction) using either uncorrelated or correlated mutations. Additionally, MOES provides several options for performing sexual reproduction using crossover [23]:

- No crossover
- Uniform crossover
- Diagonal crossover

When crossover is employed, one must choose both the recombination mode (sexual or panmictic) and the recombination operator (discrete or intermediate) independently for each class of physical and strategy parameters: the real and/or integer physical parameters, the associated standard deviations and/or mutation probabilities, as well as the covariant angles if correlated mutations are used [23]. When recombining in the sexual mode each child is produced from two parents only; when recombining in the panmictic mode, the first parent is chosen at random and then a new partner is chosen for each parameter [23]. When employing a discrete recombination operator, the child's parameter is chosen to be the corresponding parameter of one of the parents; when employing an intermediate recombination operator, the child's parameter is chosen to be the average of the parents parameters [23]. According to Bäck [25], the default modes for Evolution strategies are sexual for the physical parameters and panmictic for the strategy

parameters; the default operators are discrete for the physical parameters and intermediate for the strategy parameters.

Since each physical parameter has one associated strategy parameter, the total number of parameters is $2 \cdot [N_{real} + N_{integer}]$, where N_{real} is the number of real physical parameters and $N_{integer}$ is the number of integer physical parameters [23]. Each strategy parameter dictates how large of a step-size is allowed for a mutation of its corresponding physical parameter. In MOES, the strategy parameters for real physical parameters are standard-deviations. A high (low) standard-deviation for a given strategy parameter means larger (smaller) Gaussian perturbations are allowed on the corresponding physical parameter by the mutation operator. In effect, with high standard-deviations ES explores more of parameter space, while with low standard-deviations ES explores smaller regions of parameter space. In ES, the indication that an evolution is converging on a local optima is when the standard deviations for all the physical parameters are driven towards 0 [23].

MOES employs a common notation used in ES to describe particular "strategy". In a (μ, κ, λ)-evolution, μ is the number of parents, λ is the number of children, and κ is the maximum lifetime (in generations) of a solution [23]. When combined with truncation selection, the $\mu + \lambda$ solutions in the evolving population are ordered according to their objective values. Those solutions who have reached their maximum lifetime κ are put in the back of the queue, and the best μ of the remaining solutions are selected as the parents for the next generation [23].

In this research, MOES is used for single-objective optimization of a problem. At a high-level, the single objective optimization employed by MOES begins by using ES to evolve a population of solutions. Evolution continues until the "stall test" indicates evolution has stalled. Finally, based on the problem, either a gradient-based or non-gradient based local search is performed on

the elite population. Figure 24 illustrates the hybrid algorithm for single objective optimization employed by MOES.

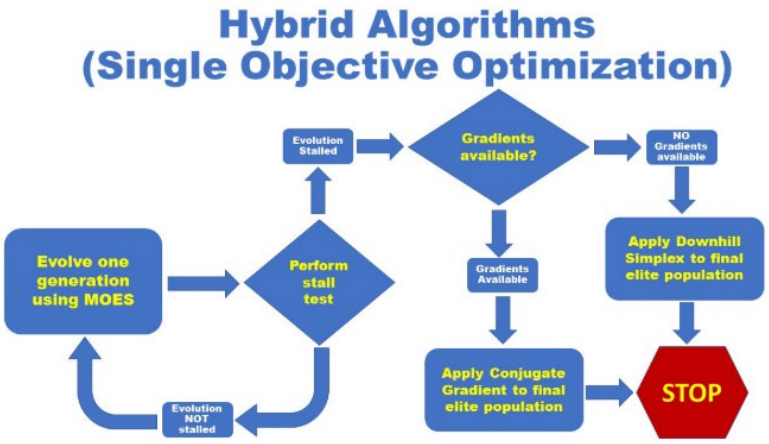


Figure 24: Hybrid algorithm for single objective optimization. Reproduced from Lill and Smith [23].

MOES, being highly customizable, offers many configurable hyperparameters which can be tuned for a given optimization problem. Table 8 summarizes the main hyperparameters of MOES.

Parameter Name	Parameter Settings	Description
Problem_Type	0, 1, or 2	0 for multiple objective, 1 for constrained single objective, and 2 for unconstrained single objective.
Num_Evolutions	$i \in \mathbb{N}$	Number of evolutions to perform.
Num_Generations	$i \in \mathbb{N}$	Maximum number of generations in a given evolution.
μ	$i \in \mathbb{N}$	Number of parents in crossover.
λ	$i \in \mathbb{N}$	Number of children in crossover.
κ	$i \in \mathbb{N}$	Maximum lifetime, in generations in crossover.
ρ	$i \in \mathbb{N}$	Number of parents used in crossover.
Selection	{truncation, tournament}	Selection mechanism employed.

Recombination	{None, uniform, diagonal}	Type of crossover to perform.
Use_Amoeba	0, 1, or 2	Apply Downhill Simplex algorithm after each evolution (1), to best solution obtained after all evolutions (2), do not apply (0).

Table 8: Main hyperparameters available for MOES. Adapted from Lill and Smith [23].

2.6 Fitness Landscapes

Fitness landscapes are a geographic metaphor used to describe the topology of a search space. A search operator traverses through a fitness landscape seeking to find the *highest* peak or *lowest* trough (for maximization and minimization problems, respectively). For some fitness landscapes coupled with a search operator, traversal towards a global optima is straightforward (e.g. the landscape is convex and selecting improving neighbors always leads towards the global optima). In others, local optima prevent the search process from reaching a global optima (e.g. the landscape is rugged, with many local optima). In such a case, the local optima "confuse" the search operator, since to the operator, it appears that every surrounding point is uphill (for minimization problem).

Metaphors aside, fitness landscapes are a useful construct to consider when devising search algorithms. Insight into the topology of the landscape can provide optimization algorithm designers with useful knowledge needed to effectively prune a search space for a high quality optima. A fitness landscape, as defined by Malan and Engelbrecht [26], includes the following:

1. A set X of configurations (solutions to the problem),
2. a notion N of neighborhood, nearness, distance, or accessibility on X, and
3. a fitness function $f : X \to \mathbb{R}$.

(1) describes the necessity of a search space in defining a fitness landscape. Next, (2) is needed to provide a relationship between elements of the search space. Finally, (3) is needed to

assess the quality of solutions in the search space. For instance, consider an optimization problem where the goal is to reach the highest point in the state of Colorado. X could be, for example, all $c_i = (longitude, latitude)$ coordinates in Colorado. N could be the Euclidean distance between two arbitrary coordinates c_1 and c_2. Finally, f could be the altitude of point c_i. Collectively, X, N, and f define a fitness landscape.

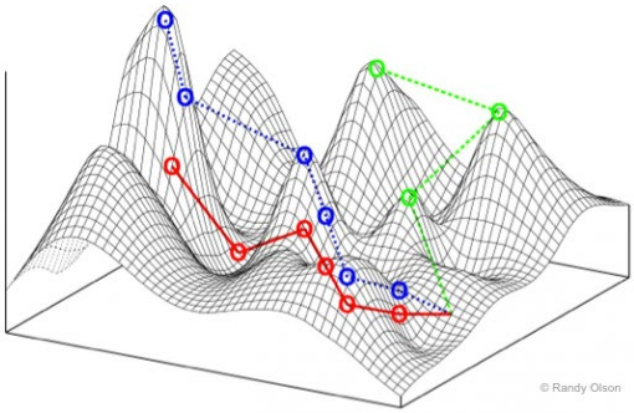


Figure 25: Example fitness landscape. From the metaphor above, all (x, y) coordinates represent coordinates in the state of Colorado. Then, N could be used to describe the distance between any two arbitrary points. Finally, f is the z axis, where z_i denotes the altitude of point (x_i, y_i). A search process could take small steps through this landscape in attempt to find the global optima. Reproduced from Olson [27].

2.6.1 Fitness Landscape Analysis Metrics

Characterizing a fitness landscape is non-trivial when the parameter space is greater than two dimensions and/or when the search space cannot be enumerated completely. In this section, a collection of fitness landscape analysis metrics is provided. Many of the techniques are from a survey of fitness landscape analysis techniques by Malan and Engelbrecht [26]. These metrics are used to gain insight into the topology of a complex fitness landscape and to analyze the effectiveness of a search operator in the landscape. Table 9 provides commonly used metrics for fitness landscape analysis.

Metric	Description
Ruggedness	This metric is used to analyze the number and distribution of local optima in the search space. Rugged landscapes have many local optima, while smooth landscapes have few local optima.
Evolvability	This metric broadly measures the capability of a search process to move to a place in the fitness landscape of better fitness.
Epistasis	This metric is used to analyze the amount of interactions among the decision variables. The more rugged the fitness landscape, the higher the epistasis [6]. Thus, ruggedness metrics can also be used to analyze epistasis.
Fitness Distribution	This metric is used to analyze how fitness values are distributed across the search space.

Table 9: Metrics commonly used to analyze fitness landscapes.

Ruggedness and Epistasis: Most commonly, ruggedness and epistasis are measured simultaneously via the autocorrelation function proposed by Weinberger [28]. The autocorrelation function, as provided in Talbi [6], is defined as follows:

$$\rho(d) = \sum_{s,t \in S \times S, dist(s,t)=d} \frac{\left(f(s) - \bar{f}\right)\left(f(t) - \bar{f}\right)}{n \cdot \sigma_f^2}$$

In this definition, d is the distance between solutions in the search space. Commonly, $\rho(1)$ is analyzed to measure ruggedness and epistasis (that is, immediate neighbors of s are analyzed). $\rho(d) \rightarrow i : i \in [-1,1]$. When $|\rho(1)| \approx 0$, this indicates the landscape is rugged (i.e. neighboring solutions have very different fitness); when $|\rho(1)| \approx 1$, this indicates the landscape is flat (i.e. neighboring solutions have similar fitness). The autocorrelation function by Weinberger has been adapted into pseudocode, as shown below in Algorithm 12.

Algorithm Autocorrelation function.

Inputs: The number of sample points n and objective function f.

X = $generate$ n random sample points in search space S;

f_s = [] ; f_t = []; /* Lists to hold scores of parents and scores of best neighbors of parents, respectively */

For x in X **Do**
  x' = $select$ a random neighbor of x ;
  $f_x = f(x)$; $f_x' = f(x')$; /* score x and neighbor x' of x, respectively */
  $f_s.append(f_x)$; $f_t.append(f_x')$; /* persist score of parent and best neighbor */
End For

$\bar{f} = mean(f_s + f_t)$; /* calculate average fitness of sample points */
/* note that $f_s + f_t$ denotes extending list f_s with list f_t */

σ_f = $standard\_deviation(f_s + f_t)$; /* calculate standard deviation of sample points */

/* Calculate $p(1)$ */
sum = 0 ;
For (i = 0 ; $i < |f_s|$; i++) **Do**
  $sum = sum + (f_s[i] - \bar{f}) * (f_t[i] - \bar{f})$;
End For

p = sum / $(n * \sigma_f^2)$;

Output p, the autocorrelation function value.

Algorithm 12: Autocorrelation function. Adapted from Weinberger [28].

Evolvability: Evolvability captures a search process's ability to move to regions of better fitness in the search space. When evolvability is high, a search process can efficiently traverse to higher-fitness regions of the search space. Conversely, when evolvability is low, moving to higher-fit regions of the search space is challenging for a search process. Evolvability can be measured via technique 16 of Malan and Engelbrecht [26] called the "Fitness Cloud" technique (devised by Verel et al. [29]). The fitness cloud technique describes the relationship between the fitness of a solution s and the fitness of its best neighbor $n^* \in N(s)$. If for many solutions in the search space neighboring solutions have higher fitness, the search process can iteratively choose improving neighbors to reach highly fit optima. Technique 16 is provided in Algorithm 13.

Algorithm Fitness cloud.

Inputs: The number of sample points n and objective function f.

X = $generate$ n random sample points in search space S;

For x in X **Do**
  x' = $select$ best neighbor $x' \in S$ of x based on search operator ;
  $plot$ the point $\big(f(x), f(x')\big)$;
End For

Algorithm 13: Fitness Cloud evolvability technique. Adapted from Malan and Engelbrecht [26].

In addition, techniques 21 and 22 of Malan and Engelbrecht (both devised by Lu et al. [30]), called the "Fitness-Probability Cloud" and "Accumulated Escape Probability" techniques, can be used to measure evolvability. The fitness-probability cloud technique is similar to the fitness cloud technique, except instead describes the relationship between the fitness of a solution s and the percentage of neighbors $N(s)$ with better fitness than s. The accumulated escape probability technique provides the probability that a random solution has neighbors of better fitness. If the escape probability is low, the search process can get "stuck" at local optima easily. Conversely, if the escape probability is high, the search process can continuously select improving neighbors to "escape" local optima. As both these techniques work in conjunction, both are provided below in Algorithm 14.

Algorithm Fitness-probability cloud with accumulated escape probability.

Inputs: The number of sample points n and objective function f.

X = $generate$ n random sample points in search space S;
P = [] ; /* List to hold P_is each iteration */

For x in X **Do**
  $neighbors$ = $find$ all neighbors $x' \in S$ of x based on search operator ;
  $neighbors\_fitness$ = $calculate$ fitness of all neighbors of x ;
  P_i = $calculate$ proportion of neighbors with greater fitness than x ;
  $plot$ the point $(f(x), P_i)$;
  $P.append(P_i)$;
End For

$\overline{P_i}$ = $mean(P)$; /* Calculate the mean of P */

Output $\overline{P_i}$, the accumulated escape probability.

Algorithm 14: Fitness-Probability Cloud evolvability technique combined with Accumulated Escape Probability technique. Adapted from Malan and Engelbrecht [26].

Fitness Distribution: Fitness distribution describes the spread of fitness scores of solutions in the search space. Statistical analysis can be performed on a fitness distribution, such as looking at the mean, mode, median, and range, to learn more about fitness spread. In order capture fitness distribution, technique 9 of Malan and Engelbrecht [26] (devised by Rose et al. [31]) can be used. Technique 9 is presented in Algorithm 15.

Algorithm Density of states.

Inputs: The number of sample points n and objective function f.

X = $generate$ n random sample points in search space S;

$scores$ = [] ; /* list to hold all observed fitness scores */
For x in X **Do**
  f_x = $f(x)$; /* Score x */
  $scores.append(f_x)$;
End For

/* statistically analyze fitness scores in $scores$ */

Algorithm 15: Density of States fitness distribution technique. Adapted from Malan and Engelbrecht [26].

2.6.2 Multi-Dimensional Scaling

If the parameter space of an optimization problem is two-dimensional, a three-dimensional graph can be used to plot the fitness landscape. Alternatively, a two-dimensional plot with colors denoting heights (commonly referred to as a *heat map*) can be used. However, for more than two-dimensions of parameter space, this visualization procedure does not work. To address this dimensionality issue, consider multidimensional scaling (MDS).

MDS is used to analyze a dissimilarity matrix and produce a spatial configuration where the objects are points [32]. According to Wickelmaier, MDS arranges the points in such a way that "their distances correspond to the similarities of the objects: similar object are represented by points that are close to each other, dissimilar objects by points that are far apart" [32]. Typically, a dissimilarity matrix is provided as input to MDS. A dissimilarity matrix is a $t \times t$ matrix D where D_{ij} represents the distance (Euclidean, Manhattan, etc.) between points i and j.

MDS is an optimization problem. Dimensionality reduction inevitably incurs some loss of information, so MDS seeks to find the best solution possible. That is, there is no way to project points from a higher dimensional space to a lower dimensional space and preserve all properties of the points in the higher dimensional space. A high quality solution of MDS is a configuration of points where distance between the points in the lower dimensional space (typically two) is as similar as possible to the distance between the points in original dimensionality of the space.

Consider a distance matrix $D^{(X)}$, which is a $t \times t$ dimensional matrix. MDS tries to find t points $y_1, \ldots, y_t$ which exist in d dimensional space. If $d_{ij}^{(X)}$ represents the distance between x_i and x_j in the original space, and $d_{ij}^{(Y)}$ denotes the distance between y_i and y_j in the d dimensional space, MDS attempts to minimize the following equation [33]:

$$MINIMIZE \sum_{i=1}^{t} \sum_{i=1}^{t} \left(d_{ij}^{(X)} - d_{ij}^{(Y)} \right)^2$$

The term being minimized is often called “stress”. When stress is minimized, points which were close in the original space are placed as close as possible in the reduced dimensional space.

2.7 Token-Swapping Problem

An important problem related to the QLP is the token-swapping problem (TokSP). Yamanaka et al. describe the TokSP as follows: “[G]iven a graph in which each vertex has an initial and target color. Each pair of adjacent vertices can [SWAP] their current colors. [The] goal is to perform the minimum number of SWAPs so that the current and target colors agree at each vertex” [34]. Consider Figure 26. In this figure, instead of each vertex having an initial and target “color”, each vertex has an initial and final token t_i.

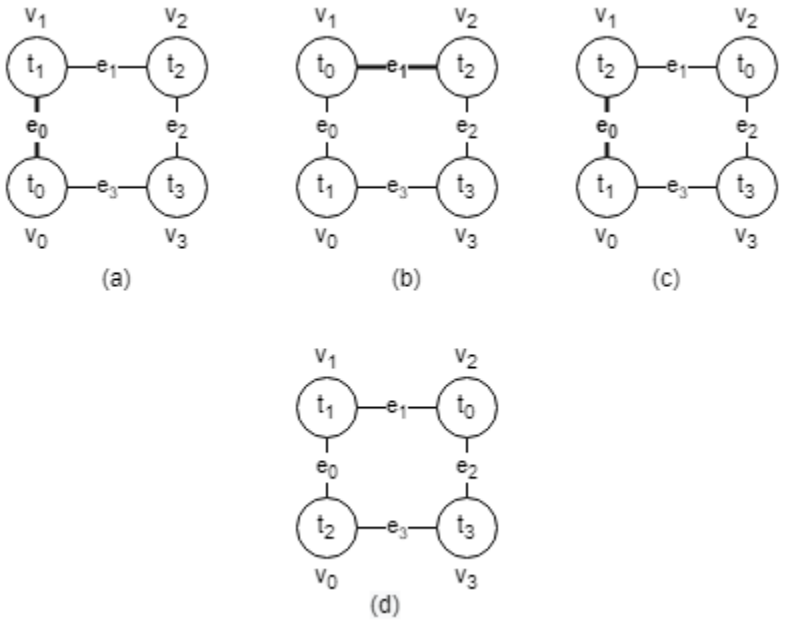


Figure 26: An instance of the token-swapping problem. Tokens on vertices are written inside circles. Vertices are labeled $v_0 \ldots v_3$ and edges are labeled $e_0 \ldots e_3$. Along each thick edge $e_i = (v_j, v_k)$, the token on v_j and v_k are swapped. (a) The initial token-placement. (b)-(c) Intermediate token-placements. (d) The target token-placement.

Alternatively, let $G = (V, E)$ define a graph with vertex-set $V = \{v_0, v_1, \dots\}$ and edge-set E and $T = \{t_0, t_1, \dots\}$ be a set of tokens that can be placed on vertices. Given an initial configuration of tokens-on-vertices f_i and a target configuration f_t, the objective of the TokSP is to find a minimum sequence of transpositions along $E(G)$ to permute $f_i \rightarrow f_t$. In general, it is difficult to find optimal solutions to the TokSP. This is supported by Miltzow et al. who prove that the TokSP is NP-complete [35].

Due to the NP-hardness of the TokSP, finding optimal solutions to TokSP problem-instances is a computationally intractable task. Nonetheless, Miltzow et al. devised an approximation algorithm for TokSP that returns a solution no worse than $4 \cdot OPT$, where OPT is the cost of the optimal solution. Their approximation algorithm is guaranteed to run in polynomial time, unlike exact methods which run in exponential time. A refinement of their approximation algorithm for the TokSP is presented in Appendix B.

Miltzow et al. also provide lower- and upper-bounds for the TokSP. In their paper, they begin by defining function $d(t_i)$ which yields the distance of token t_i to its target vertex v_t. Then, they define the aggregate distance D of all tokens from their initial vertices to final vertices as follows:

$$D = \sum_{t_i \in T} d(t_i)$$

They then prove that their 4-approximation algorithm requires at least $\frac{D}{2}$ SWAPs and at most $4D$ SWAPs [35].

2.8 Summary

Section 2.2 begins this chapter by providing background information on an alternative form of computation: quantum computing. Next, the necessity for and steps taken in QPT are presented in Section 2.3. In the QPT background, the QLP is defined as a critical subroutine of QPT. Thereafter, SOTA QLP-solvers are presented and analyzed. Then Section 2.4 provides a tour of a

quantum computing framework called Qiskit. A variety of metaheuristic algorithms used in this research effort are defined in Section 2.5 along with fitness landscape analysis procedures in Section 2.6. Finally, the TokSP is presented in Section 2.7.

III. Methodology

3.1 Overview

This chapter defines and describes the methodology used to analyze the research questions posed in Chapter I. First, an overview of the approach of this research is provided in Section 3.2. In Section 3.3, the QLP is mathematically modeled and integrated into various metaheuristic algorithm domains. Section 3.3 also justifies the applicability of metaheuristics to finding solutions to the QLP. Section 3.4 outlines the performance analysis metrics used in this research to compare the devised meta-based QLP-solvers versus SOTA QLP-solvers. Finally, Section 3.5 provides procedures to analyze the fitness landscape of the QLP.

3.2 Approach

This research effort has three primary goals. The first is to develop meta-based QLP-solvers. The second is to analyze the performance of the meta-based QLP-solvers versus SOTA QLP-solvers. The final goal is to analyze the fitness landscape of the QLP.

3.3 Design of QLP-Solvers using Metaheuristics

Table 10 provides steps to design metaheuristic algorithms for optimization problems per recommendations of Talbi [6].

Steps to Design Metaheuristic Algorithms for Optimization Problems	
1	Model the problem
2	Analyze the complexity/difficulty of the problem
3	Analyze the requirements of the application
4	Design a metaheuristic
5	Tune parameters
6	Evaluate performance

Table 10: Metaheuristic algorithm design steps. Adapted from Talbi [6].

In Section 3.3, Steps 1 through 5 are taken to design metaheuristic algorithms to find high-quality solutions to the QLP. In Section 3.4, Step 6 is addressed, in which test procedures and metrics are defined for performance analysis of the devised QLP-solver.

3.3.1 Model the Problem

Integer programming (IP) is used to model the QLP as an optimization problem. In the design of metaheuristic algorithms, the identification of decision variables, constraints, and objective function(s) is required to model the problem as an optimization problem that metaheuristics can optimize. When modeling a problem, Talbi suggests developing a model based on others proposed in previous literature. In Section 2.3.2, an abstract solution M_h to the QLP is defined as follows:

$$M_h = (\lambda_0, \pi_1, \lambda_1, \pi_2, \dots, \lambda_{L-2}, \pi_{L-1}, \lambda_{L-1}) : \lambda_i \vdash \mathrm{l_i}$$

where L is the number of layers in the circuit.

Reiterating, to find solutions to the QLP, first a set of satisfying layouts $\lambda_0 \dots \lambda_{L-1}$ must be found for each l_i of the circuit. Then, based on the $\lambda_i's$, SWAP layers $\pi_1 \dots \pi_{L-1}$ must be found, where each π_i permutes λ_{i-1} to λ_i. The goal then of the QLP is to find the best M_h, in the sense

of having the highest probability of executing the problem-instance circuit successfully on the problem-instance backend. This definition directly follows from the works of Tannu and Qureshi [4] and Murali et al. [12], and it corresponds to the second definition of the QLP in Section 2.3.2.

The following sections cast the QLP as an IP problem based on the model defined by Zulehner, Paler, and Wille [11] with the objective defined by Tannu and Qureshi and Murali et al. Section 3.3.1.1 defines mathematical formalisms needed for the IP formulation of the QLP. Section 3.3.1.2 enumerates the decision variables for the QLP. Section 3.3.1.3 defines the constraints of the QLP. Finally, Section 3.3.1.4 and 3.3.1.5 construct objective functions for the QLP.

3.3.1.1 QLP Formalisms

Let M be the number of logical qubits in the circuit and N the number of physical qubits on the QC topology, $V = \{P_0, P_1, \dots, P_{N-1}\}$ the set of physical qubits, and $T = \{q_0, q_1, \dots, q_{N-1}\}$ the set of logical qubits. Note that when $M < N$, $\{q_M \dots q_{N-1}\}$ are unused logical qubits. On the other hand, if $M > N$, the circuit cannot be executed on the topology, as there are not enough physical qubits to hold all logical qubits. Let $G = (V, E)$ be a weighted directed graph defining the topology of a backend QC. The edge-weight of a directed edge $e_{ij} = (P_i, P_j) \in E(G)$ is the 2-qubit gate error rate between P_i and P_j. A P2L mapping is expressed via the bijective function $\lambda : V \to T$. To begin, first initialize each λ_i to the trivial P2L mapping. That is, initialize

$$\lambda_i = \begin{pmatrix} P_0 & P_1 & \dots & P_{N-1} \\ q_0 & q_1 & \dots & q_{N-1} \end{pmatrix}.$$

In each mapping, $\lambda_i(P_j) = q_k$ means physical qubit P_j is maps to logical qubit q_k in λ_i. Consider the following P2L mapping:

$$\lambda_i = \begin{pmatrix} P_0 & P_1 & P_2 & P_3 \\ q_0 & q_1 & q_2 & q_3 \end{pmatrix}.$$

In this example $N = 4,\ M \leq N, V = \{P_0, P_1, P_2, P_3\}$, and $T = \{q_0, q_1, q_2, q_3\}$. λ_i represents the following P2L mapping:

$$P_0 \mapsto q_0$$
$$P_1 \mapsto q_1$$
$$P_2 \mapsto q_2$$
$$P_3 \mapsto q_3$$

Equivalently stated, $\lambda_i(P_0) = q_0$, $\lambda_i(P_1) = q_1$, $\lambda_i(P_2) = q_2$, and $\lambda_i(P_3) = q_3$. Thus, all physical qubits are assigned distinct logical qubits. Since λ is a bijective function, $\lambda^{-1} : T \to V$ represents the inverse mapping.

Let $t_{jk} : T \to T$ be the transposition of logical qubits q_{i_j} and q_{i_k}:

$$t_{jk} = \begin{pmatrix} q_{i_0} \cdots q_{i_j} \cdots q_{i_k} \cdots q_{i_{N-1}} \\ q_{i_0} \cdots q_{i_k} \cdots q_{i_j} \cdots q_{i_{N-1}} \end{pmatrix}.$$

Then, given a mapping λ_i where $\lambda_i(P_j) = q_l$ and $\lambda_i(P_k) = q_m$, $\lambda_i' = t_{jk} \circ \lambda_i$ yields $\lambda_i'(P_j) = q_m$ and $\lambda_i'(P_k) = q_l$. The application of a transposition after a P2L mapping "exchanges" logical qubits between two physical qubits. Accordingly, a transposition is commonly called a 1-exchange operation. More generally, an n-exchange operation denotes the application of n transposition operations to a mapping. In this definition, n is the total number of transposition operations to apply to a given mapping and not the minimum number of transpositions to achieve the resulting permutation. For example, $t_{jk} \circ t_{jk} \circ \lambda_i = \lambda_i$ is a 2-exchange even though the resulting permutation is the same as the initial.

In M_h, π_i is a sequence of SWAP operations needed to permute configuration λ_{i-1} to λ_i. Mathematically, π_i is a sequence $(e_{ab}, e_{cd}, \ldots)$ where $e_{ab}, e_{cd} \ldots \in E(G)$. Let $\pi_{i,j}$ be the term in π_i with index j (where the first term in the sequence is $\pi_{i,0}$). Given that an edge $e_{mn} = (P_m, P_n)$, let $t(e_{mn})$ yield transposition t_{mn}. Then π_i is a satisfying SWAP sequence if:

$$t(\pi_{i,l}) \circ \dots \circ t(\pi_{i,1}) \circ t(\pi_{i,0}) \circ \lambda_{i-1} = \lambda_i$$
$$\text{where } l = |\pi_i| - 1.$$

3.3.1.2 Decision Variables

In this section, the random-key encoding presented in Section 2.4 is used to represent unique permutations of logical on physical qubits for configurations $\lambda_0 \dots \lambda_{L-1}$. As described in Section 2.4, random-key encoding uses real-valued variables to represent permutations [21]. For the QLP, encoding is performed via the function $\chi : V \to [0,1]$. First, for each $\lambda_i \in M_h$, the encoding creates an associated mapping χ_i. Decoding is then applied as follows: for each λ_i, the decoder visits each $P_0, P_1, \dots P_{N-1} \in V(G)$ and sorts (in ascending order via a stable sorting algorithm) λ_i based on the corresponding χ_i. Consider Algorithm 16, which uses a naïve stable sorting algorithm to decode unique permutations of each λ_i based on their corresponding χ_i's.

Algorithm Decoding unique permutations for each λ_i via random-key encoding.

Inputs: Decision variables $\chi_0 \dots \chi_{L-1}$, number L of layers in the circuit, graph G defining the QC topology, number N of physical qubits, and the set T of logical qubits.

For $(i = 0 ; i < L ; i{+}{+})$ **Do**

$U = V(G)$; /* current unsorted set */

/* create mapping λ_i based on χ_i */
For $(j = 0; j < N; j{+}{+})$ **Do**

$P = argmin_{y \in U}\, \chi(y)$; /* get unsorted vertices with lowest assigned value */
P_{min} = select $P_k \in P$ with lowest index k ; /* ensures stable sort */

$\lambda_i(P_{min}) = q_j$; /* add mapping */
$U = U - \{P_{min}\}$; /* remove P_{min} from unsorted set */

Output $\lambda_0 \dots \lambda_{L-1}$, sorted based on $\chi_0 \dots \chi_{L-1}$, respectively.

Algorithm 16: Random-key decoding for the QLP.

In the encoding phase, a χ_i is created for each λ_i. To construct a χ_i, each element P_j of the domain V is mapped to a real number k_{ij} selected randomly according to a continuous uniform

distribution over [0,1]. Since the domain of χ has cardinality N, and L distinct χ_is are created during encoding, this encoding results in $L \cdot N$ real-valued decision variables for the QLP, enumerable as follows:

$$x_{ij} = k_{ij} : k_{ij} \sim U([0,1])$$
$$\forall i \in [\![0, L-1]\!], \forall j \in [\![0, N-1]\!].$$

Consider Figure 27.

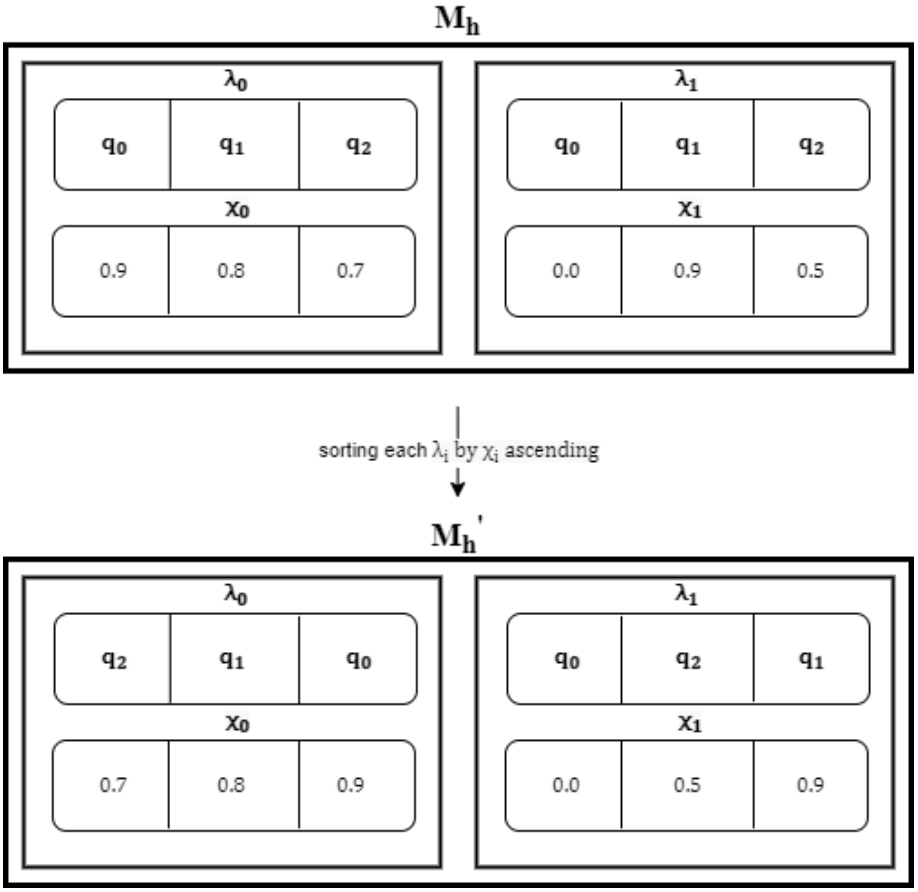


Figure 27: Random-key encoding example for the QLP.

The QLP problem-instance in Figure 27 considers a circuit with $M = 3$ logical qubits and $L = 2$ layers, and a QC with $N = 3$ physical qubits. First, two P2L mappings are created for each layer, each of which consist of three elements (one for each physical qubit). Then, a χ_i is created for

each λ_i. The previous two steps represent the encoding phase. Finally, to decode a valid permutation of each λ_i, each λ_i is sorted by its corresponding χ_i.

3.3.1.3 Constraints

At first glance, the QLP appears to be a constrained optimization problem due to the connectivity constraints of the physical topology. That is, C_{NOT} operations can only be executed if all control, target pairs (q_c, q_t) are placed in physically adjacent physical qubits (P_γ, P_τ). However, these constraints can be eliminated from the IP formulation of the QLP by considering the generalized bridge operation (GBO).

Generalized Bridge Operation: Given a logical control qubit q_c placed in physical qubit P_γ and a logical target qubit q_t placed in physical qubit P_τ, a bridge operation allows one to achieve the effect of a C_{NOT} operation between q_c and q_t when the distance between P_γ and P_τ is greater than one link. Moreover, performing the bridge operation does not permute the logical qubits on physical qubits, and none of the states of the intermediate qubits between q_c and q_t are altered. Accordingly, the GBO is a layout-preserving operation since the GBO does not induce SWAP operations.

In some cases, performing a bridge as opposed to swapping requires fewer C_{NOT} operations, and in others, the bridge incurs a large cost. The cost of a bridge operation depends on the distance separating P_γ and P_τ. For instance, if the distance between P_γ and P_τ in G is two, a bridge operation at this distance requires four C_{NOT} operations, whereas swapping at this distance requires two SWAP operations + one C_{NOT} = seven C_{NOT}s (as each SWAP gate decomposes to three C_{NOT}'s and after swapping the requested C_{NOT} gate is performed). Even though the bridge operation in this example requires fewer C_{NOT} operations, if the SWAP operations reduce the distance between other qubits involved in C_{NOT} operations, swapping can be more beneficial than

bridging in the previous example. The objective functions proposed below rate such candidates appropriately.

Siraichi et al. provide an example bridge operation when the distance between the physical qubits holding q_c and q_t is 2 [13]. Figure 28 illustrates how such a bridge operation is performed.

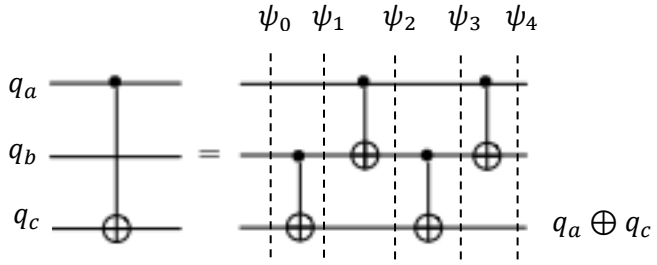


Figure 28: Bridge operation at distance 2 where q_a is the control qubit, q_c is the target qubit, and q_b is an intermediate qubit between q_a and q_c. Adapted from Siraichi et al. [13].

Next, quantum state analysis [7] is used to demonstrate the correctness of the bridge operation posed in Figure 28. Let $\psi_0 = |q_a q_b q_c\rangle$. Because $C_{NOT}|q_\alpha, q_\beta\rangle = |q_\alpha, q_\alpha \oplus q_\beta\rangle$, it must be shown that $|\psi_4\rangle = |q_a, q_b, q_a \oplus q_c\rangle$. Let $C_{NOT}(q_i, q_j)$ denote a C_{NOT} operation between control q_i and target q_j.

$$
\begin{aligned}
&|\psi_0\rangle = |q_a\, q_b\, q_c\rangle \\
&|\psi_1\rangle = C_{NOT}(q_b, q_c)|\psi_0\rangle = |q_a, q_b, (q_b \oplus q_c)\rangle \\
&|\psi_2\rangle = C_{NOT}(q_a, q_b)|\psi_1\rangle = |q_a, (q_a \oplus q_b), (q_b \oplus q_c)\rangle \\
&|\psi_3\rangle = C_{NOT}(q_b, q_c)|\psi_2\rangle = |q_a, (q_a \oplus q_b), [(q_a \oplus q_b) \oplus (q_b \oplus q_c)]\rangle \\
&|\psi_4\rangle = C_{NOT}(q_a, q_b)|\psi_3\rangle = |q_a, [q_a \oplus (q_a \oplus q_b)], [(q_a \oplus q_b) \oplus (q_b \oplus q_c)]\rangle
\end{aligned}
$$

The properties shown in Table 11 can be used to reduce $|\psi_4\rangle$.

XOR Properties	
Identity	$A \oplus 0 = A$
Self Inverse	$A \oplus A = 0$
Communitive	$A \oplus B = B \oplus A$
Associative	$A \oplus (B \oplus C) = (A \oplus B) \oplus C$

Table 11: XOR Properties. Adapted from Lewin [36].

$$
\begin{aligned}
|\psi_4\rangle &= |q_{a,}\,[q_a \oplus (q_a \oplus q_b)], [(q_a \oplus q_b) \oplus (q_b \oplus q_c)]\rangle \\
&= \left|q_{a,}\,[(q_a \oplus q_a) \oplus q_b], [q_a \oplus (q_b \oplus q_b) \oplus q_c]\right\rangle \\
&= \left|q_{a,}\,(0 \oplus q_b), (q_a \oplus 0 \oplus q_c)\right\rangle \\
&= |q_a, q_b, (q_a \oplus q_c)\rangle
\end{aligned}
$$

Thus, only the state of the target qubit is changed via the bridge operation at distance 2. No other qubits are affected. Algorithm 17 presents a novel algorithm to perform a bridge operation at an arbitrary distance.

Algorithm Generalized Bridge Operation.

Preliminaries: Let q_c and q_t be the control and target logical qubits of a C_{NOT} operation, respectively. Let P_γ and P_τ be the physical qubits that hold q_c and q_t, respectively. Let p be a sequence holding the shortest weighted path between P_γ and P_τ in G (where the edge weights are the 2-qubit error rates between adjacent physical qubits. The elements of p are unique physical qubits from $V(G)$. Further, $p_0 = P_\gamma$ and $p_d = P_\tau$.). Let $d = |p| - 1$ (where $|p|$ denotes the number of vertices in p).

C_{NOTs} = [] ; /* List to hold C_{NOT} ops to perform, defined by (P_i, P_j) pairs where P_i is the control and P_j is the target */

/* Phase ρ_1 : descending C_{NOT} operations */
/* "Descending" refers to moving backwards along path p */
For ($i = d$; $i > 0$; i--) **Do**
  $C_{NOTs}.append((p_{i-1},\ p_i))$;
End For

/* Phase ρ_2 : ascending C_{NOT} operations */
/* "Ascending" refers to moving forwards along path p */
For (i = 1 ; $i < d$; i++) **Do**
  $C_{NOTs}.append((p_i,\ p_{i+1}))$;
End For

/* Phase ρ_3 : repair */
For (i = 1 ; $i \leq (2 \cdot d) - 3$; i++) **Do**
  $C_{NOTs}.append(C_{NOTs}[i])$;
End For

Output C_{NOTs}, which is a sequence of (P_i, P_j) pairs which must be performed, in order, to simulate $C_{NOT}(P_c, P_t)$.

Algorithm 17: Generalized bridge operation.

This algorithm consists of phases ρ_1, ρ_2, and ρ_3. Let $|p_i\rangle$ denote the state of qubit p_i and let $|\psi_0\rangle = |p_0 p_1 \dots p_d\rangle$ define the initial state of the system. In phase ρ_1, a descending chain of C_{NOT} operations is performed to transform the initial state of the system $|\psi_0\rangle \rightarrow |\psi_{\rho_1}\rangle$, where $|\psi_{\rho_1}\rangle = |p_0\rangle(\otimes_{i=1}^{d} |p_{i-1} \oplus p_i\rangle)$. Next, phase ρ_2 performs an ascending chain of C_{NOT} operations to transform the state of the system $|\psi_{\rho_1}\rangle \rightarrow |\psi_{\rho_2}\rangle$, where $|\psi_{\rho_2}\rangle = |p_0\rangle(\otimes_{i=1}^{d} |p_0 \oplus p_i\rangle)$. The resultant state of $|\psi_{\rho_2}\rangle$ occurs due to XOR cancellations. After phase ρ_2 completes, p_0 and p_d are entangled. However, all qubits along p are also entangled with p_0. To fix this, phase ρ_3 performs the same sequence of C_{NOT}s that were performed to get the intermediate qubits to the states $|p_0 \oplus p_i\rangle$. Since the C_{NOT} is its own inverse operation, applying the same sequence of C_{NOT}s reverts the intermediate qubits back to their original states. After ρ_3, the state of the system is transformed from $|\psi_{\rho_2}\rangle \rightarrow |\psi_{\rho_3}\rangle$, where $|\psi_{\rho_3}\rangle = |p_0 p_1 \dots p_{d-1}\rangle|p_0 \oplus p_d\rangle$. Thus, this algorithm successfully entangles the distant control and target qubits without disrupting the states of other qubits (and requires no SWAP operations). Figure 29 illustrates the phases of the GBO.

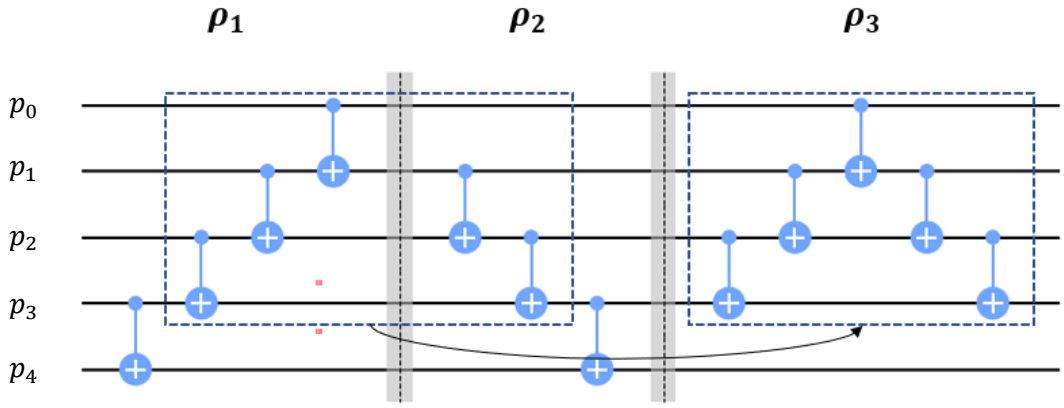


Figure 29: Illustration of GBO where p_0 is the control and p_4 is the target, and the distance between them is 4. In ρ_1, a chain of descending C_{NOT}s is performed. In ρ_2, a chain of ascending C_{NOT}s is performed. Finally, in ρ_3, the portion of the circuit that got p_1, p_2, and p_3 entangled with p_0 is re-executed to return them to their original states.

3.3.1.4 Objective Functions

After decision variables and constraints are defined, the final step in modeling the QLP as an optimization problem is to define an objective function. The objective function encapsulates the quality of a solution to the QLP. A high-quality solution to the QLP contains P2L mappings that:

- place logical qubits involved in 1-qubit operations in highly-reliable physical qubits;
- place pairs of logical qubits involved in 2-qubit operations in physical qubits that have highly-reliable links between them; and
- are as uniform as possible, meaning the mappings induce SWAP layers $\pi_1 \ldots \pi_{L-1}$ of minimum aggregate cardinality. That is, $\sum_{i=1}^{L-1}|\pi_i|$ is minimized.

The goal is to find high-quality P2L mappings that meet these three criteria, and as such they motivate the objective functions:

1. Objective 1. Maximize the probability of successful gate execution for both 1- and 2-qubit operations.
2. Objective 2. Maximize the probability of successfully swapping logical qubits between each λ_i and λ_{i+1}.

The first two criteria seek to maximize the reliability of each layout by placing logical qubits in physical qubits that are highly-reliable and connected by highly reliable links, leading to Objective 1. The third criterion seeks to minimize the number of SWAP operations needed to permute between layouts. Following prior research efforts by Murali, et al. [12] as well as Tannu and Qureshi [4], Objective 2 is reformulated to maximize the probability of successfully executing all SWAP operations.

Note that while these objective functions do not explicitly use the decision variables, each λ_i implicitly uses them. Let a partial solution s to the QLP be defined as follows:

$$s = (\lambda_0, \lambda_1, \dots, \lambda_{L-1})$$

where L is the number of layers in the circuit.

For $i \in [\![1, L-1]\!]$ each λ_i in s has been appropriately permuted relative to λ_{i-1} via the decision variables. Partial solution s is used because the calculation of $\pi_1 \dots \pi_{L-1}$ is incorporated into the objective function. As such, their values are determined by the selected P2L mappings.

Objective 1 Formulation: Objective 1 seeks to maximize the probability of successfully executing all 1-qubit and 2-qubit gate operations within each layout based on the assigned P2L mappings for each λ_i. In other words, the goal is to minimize the cost of placing each logical qubit q_i in physical qubit P_j (in terms of both 1-qubit and 2-qubit operations) per layer.

To begin, the sub-objective for 1-qubit gate success probability is defined. First, let $r_1(P_i)$ be the probability that a 1-qubit operation executes successfully on P_i. Next, let $g_1(l_i, q_j) = 1$ if a 1-qubit gate is assigned to q_j in l_i; 0 otherwise. Then the probability that a 1-qubit operation executes successfully on q_j, placed in physical qubit $P_k = \lambda_i^{-1}(q_j)$, is

$$s_1(\lambda_i, q_j) = \begin{cases} r_1\left(\lambda_i^{-1}(q_j)\right) \text{ if } g_1(l_i, q_j) = 1, \\ 1 \text{ otherwise.} \end{cases} \tag{3.1}$$

Each 1-qubit operation in l_i is treated as an independent event, and all such events must be successful for the event that all 1-qubit operation in l_i execute without error to occur. Accordingly, for a given layout λ_i and associated layer l_i, the probability $\xi_1(\lambda_i)$ that all 1-qubit operations execute successfully in l_i is the product of the 1-qubit success rates of all physical qubits that hold a logical qubit assigned to a 1-qubit gate in l_i. This probability is

$$\xi_1(\lambda_i) = \prod_{q_j \in T} s_1(\lambda_i, q_j) \tag{3.2}$$

Now, given a solution $s = (\lambda_0, \lambda_1, \ldots, \lambda_{L-1})$ to the QLP, the probability $f_{1q}(s)$ that all 1-qubit operations execute successfully is the product of $\xi_1(\lambda_0) \cdot \xi_1(\lambda_1) \cdot \ldots \cdot \xi_1(\lambda_{L-1})$ because all 1-qubit operations of each layer are assumed to be independent events, and all events must be successful for the event that all 1-qubit operations execute without error to occur. This probability is

$$MAXIMIZE\ f_{1q}(s) = \prod_{\lambda_i \in s} \xi_1(\lambda_i) \tag{3.3}$$

where s is a partial solution to the QLP.

Next, the sub-objective for 2-qubit gate success probability is defined. First, let $e(P_i, P_j) = 1$ if there is a directed edge from P_i to P_j in G; 0 otherwise. Second, let $r_2(P_i, P_j)$ be the probability that a 2-qubit operation executes successfully between control P_i and target P_j if $e(P_i, P_j) = 1$; 0 otherwise. Third, let $g_2(l_i, q_j, q_k) = 1$ if a 2-qubit gate is assigned between control qubit q_j and target qubit q_k in l_i; 0 otherwise. Fourth, let $p = sp(P_i, P_j)$ be the shortest weighted path from P_i to P_j (see Section 1.5 for conventions).

Now, given a 2-qubit operation (for IBM's QCs, all 2-qubit operations decompose to C_{NOT} operations) between control q_a placed in P_i and target q_b placed in P_j, the probability that the gate executes successfully depends on the distance separating P_i and P_j and the direction of the link connecting them in G.

1. In the *trivial case*, there is a directed edge $P_i \rightarrow P_j$. The probability of the 2-qubit operation executing successfully is the fidelity of the link $P_i \rightarrow P_j$.
2. In the *backwards case*, there is only a directed edge $P_j \rightarrow P_i$. A reversal operation is performed to apply the C_{NOT} operation along edge $P_j \rightarrow P_i$ (as shown in Figure 7) since

P_j is the target (needs to be the control) and P_i is the control (needs to be the target). A reversal operation requires two Hadamard operations on both P_i and P_j and a C_{NOT} between control P_j and target P_i. Since all operations of a reversal are assumed to be indepependent events, and and all events must be successful for the event that a reversal operation executes without error to occur, the probability of the reversal operation executing successfully is the fidelity of the link $P_j \rightarrow P_i$ multiplied by the fidelity of applying two Hadamard gates both on P_i and P_j.

3. In the *either-direction case,* there are directed edges $P_i \rightarrow P_j$ and $P_j \rightarrow P_i$. The probability of the C_{NOT} operation executing successfully is the highest-fidelity choice of the trivial and backwards cases.
4. In the *bridge case*, P_i and P_j do not share a directed edge in either direction. The probability of the C_{NOT} operation executing successfully is the fidelity of performing a bridge operation from the control P_i to the target P_j (via the GBO of Section 3.3.1.3). First, let $p = sp(P_i, P_j)$ be the shortest weighted path from P_i to P_j, and $d = |p| - 1$. Based on results presented in Appendix A, bridging between two qubits along path p consists of the following operations:
    - Event 1: Two C_{NOT} operations must first be performed between qubits $P_a = p_0$ and $P_b = p_1$.
    - Event 2: Two C_{NOT} operations must be performed between qubits $P_a = p_{d-1}$ and $P_b = p_d$.
    - Event 3: Four C_{NOT} operations must be performed between all pairwise adjacent qubits between P_i and P_j in p.

    Let β, γ, and α be the probabilities that the operations in Events 1, 2, and 3 execute successfully, respectively. Since all operations of Events 1, 2, and 3 are assumed to be

independent events, and all events must be successful for the bridge operation to execute without error, the probability of successfully bridging between distant qubits P_i and P_j is the product of $\alpha \cdot \beta \cdot \gamma$.

Considering cases 1-4, the probability that a C_{NOT} gate between control P_i and target P_j is

$$b(P_i, P_j) = \begin{cases} r_2(P_i, P_j) \text{ if } e(P_i, P_j) = 1 \text{ and } e(P_j, P_i) = 0, \\ \omega \text{ if } e(P_j, P_i) = 1 \text{ and } e(P_i, P_j) = 0, \\ \max\{r_2(P_i, P_j), \omega\} \text{ if } e(P_i, P_j) = 1 \text{ and } e(P_j, P_i) = 1, \\ \alpha \cdot \beta \cdot \gamma \text{ if } e(P_i, P_j) = 0. \end{cases} \tag{3.4}$$

s.t.

$$\omega = r_2(P_j, P_i) \cdot \left(r_1(P_i)\right)^2 \cdot \left(r_1(P_j)\right)^2,$$
$$p = sp(P_i, P_j) \text{ and } d = |p| - 1,$$
$$\beta = \left(b(p_0, p_1)\right)^2,$$
$$\alpha = \prod_{h=1}^{d-2} \left(b(p_h, p_{h+1})\right)^4, \text{ and}$$
$$\gamma = \left(b(p_{d-1}, p_d)\right)^2.$$

If a C_{NOT} gate is assigned between control $P_l = \lambda_i^{-1}(q_j)$ and target $P_m = \lambda_i^{-1}(q_k)$ in l_i, then $b(P_l, P_m)$ is the probability the gate executes successfully. Otherwise, the probability that the gate executes successfully is 1 since no operation needs to be executed. Thus, the probability that a C_{NOT} in l_i between q_j and q_k executes successfully is

$$s_2(\lambda_i, q_j, q_k) = \begin{cases} b\left(\lambda_i^{-1}(q_j), \lambda_i^{-1}(q_k)\right) \text{ if } g_2(l_i, q_j, q_k) = 1, \\ 1 \text{ otherwise.} \end{cases} \tag{3.5}$$

For a given layout λ_i and associated layer l_i, the probability $\xi_2(\lambda_i)$ that all 2-qubit operations execute successfully in l_i is the product of the 2-qubit success rates of all physical qubit pairs that hold a logical qubit pair assigned to a 2-qubit gate in l_i. This is because each 2-qubit operation in

l_i is treated as an independent event, and all such events must be successful for the event that all 2-qubit operations in l_i execute without error to occur. This probability is

$$\xi_2(\lambda_i) = \prod_{q_j \in T} \prod_{q_k \in T} s_2(\lambda_i, q_j, q_k) \tag{3.6}$$

Now, given a solution $s = (\lambda_0, \lambda_1, \dots, \lambda_{L-1})$ to the QLP, the probability $f_{2q}(s)$ that all 2-qubit operations execute successfully is the product of $\xi_2(\lambda_0) \cdot \xi_2(\lambda_1) \cdot \dots \cdot \xi_2(\lambda_{L-1})$ because all 2-qubit operations of each layer are treated as independent events, and all events must be successful for the event that all 2-qubit operations execute without error to occur. This probability is

$$MAXIMIZE\ f_{2q}(s) = \prod_{\lambda_i \in s} \xi_2(\lambda_i) \tag{3.7}$$

where s is a partial solution to the QLP.

Finally, for the circuit to execute successfully, all 1-qubit gates must execute correctly and all 2-qubit gates must execute correctly. These events are assumed to be independent, so the probability of both occurring is

$$MAXIMIZE\ f_q(s) = f_{1q}(s) \cdot f_{2q}(s) \tag{3.8}$$

where s is a partial solution to the QLP.

Objective 2 Formulation: Objective 2 seeks to maximize the probability of successfully swapping the logical qubits among the physical qubits between each λ_i and λ_{i+1}. Such fidelity is a crucial consideration, as performing many SWAP operations highly degrades the fidelity of the overall circuit.

Finding a sequence of SWAP operations to permute between each λ_{i-1} and λ_i is an instance of the NP-Hard TokSP, which is described by Miltzow et al. [35] (see Section 2.7). No known polynomial-time algorithm exists to find minimal sequences of SWAPs to perform the necessary permutations between layouts, and such an algorithm is unlikely to exist due to the NP-hardness of the TokSP. Nonetheless, the 4-approximation algorithm devised by Miltzow et al. (presented in Appendix B) provides a computationally tractable means of finding a good sequence of SWAPs to permute between each λ_{i-1} and λ_i. Let $w(\lambda_i, \lambda_j)$ map ordered pairs of P2L mappings to sequences of edges of G, in accordance with the application of the 4-approximation algorithm with λ_i and λ_j as input.

The 4-approximation function $w(\lambda_i, \lambda_j, r)$ takes as parameters an initial configuration λ_i, a final configuration λ_j, and an optional random seed r. If a random seed is provided w acts as a deterministic function (without a seed, w is a random function). Function $w(\lambda_i, \lambda_j, r)$ yields a sequence $\pi_s = \big((P_i, P_j), (P_k, P_l), \dots\big)$ of edges from the $E(G)$. First, let $\pi_{s,j}$ denote the j^{th} element in sequence π_s. Then let $\pi_{s,j,0}$ ($\pi_{s,j,1}$) refer to the first (second) vertex in edge $\pi_{s,j}$.

Given a SWAP operation (for IBM's QCs, all SWAP operations decompose to three C_{NOT} operations; see Figure 9 of Section 2.2.3) between q_a placed in P_i and q_b placed in P_j, the probability that the SWAP gate executes successfully depends on the direction of the link connecting them in G (P_i and P_j are assumed to be adjacent in G).

- Case 1: There is only a directed edge from $P_i \rightarrow P_j$. In this case, $C_{NOT}(P_i, P_j)$ is performed twice. In addition, $C_{NOT}(P_j, P_i)$ must also be performed. Since there is no directed edge from $P_j \rightarrow P_i$, a reversal is performed to achieve $C_{NOT}(P_j, P_i)$. Accordingly, three C_{NOT}s are performed along edge (P_i, P_j) and two Hadamard operations are performed on both P_i and P_j.

- Case 2: There is only a directed edge from $P_i \rightarrow P_j$. In this case, $C_{NOT}(P_i, P_j)$ is performed twice. In addition, $C_{NOT}(P_j, P_i)$ must also be performed. Since there is no directed edge from $P_j \rightarrow P_i$, a reversal is performed to achieve $C_{NOT}(P_j, P_i)$. Accordingly, three C_{NOT}s are performed along edge (P_j, P_i) and two Hadamard operations are performed on both P_i and P_j.
- Case 3: There is an edge in both directions and the highest-fidelity SWAP is selected. No reversal operation is required for this case. Accordingly, either three C_{NOT}s are performed along edge (P_i, P_j) or three C_{NOT}s are performed along edge (P_j, P_i) (whichever direction has higher-fidelity).

Let event e be the probability that the operations of the appropriate case (Case 1, Case 2, or Case 3) execute successfully. Since all operations of the case are treated as independent events, and all events must be successful for the event that the SWAP operation executes without error to occur, the probability of successfully swapping between qubits P_i and P_j is

$$\delta(P_i, P_j) = \begin{cases} \left(r_2(P_i, P_j)\right)^3 \cdot \left(r_1(P_i)\right)^2 \cdot \left(r_1(P_j)\right)^2 & if\ e(P_i, P_j) = 1 \text{ and } e(P_j, P_i) = 0, \\ \left(r_2(P_j, P_i)\right)^3 \cdot \left(r_1(P_i)\right)^2 \cdot \left(r_1(P_j)\right)^2 & if\ e(P_i, P_j) = 1 \text{ and } e(P_i, P_j) = 0, \\ \max\{\mu, \nu\} & if\ e(P_i, P_j) = 1 \text{ and } e(P_j, P_i) = 1. \end{cases} \tag{3.9}$$

where

$$\mu = \left(r_2(P_i, P_j)\right)^2 \cdot \left(r_2(P_j, P_i)\right), \text{ and}$$
$$\nu = \left(r_2(P_j, P_i)\right)^2 \cdot \left(r_2(P_i, P_j)\right).$$

Each SWAP operation in π_s is treated as an independent event, and all such events must be successful for the event that all SWAP operations in π_s execute without error to occur. Thus, the probability that all SWAP operations in π_s execute successfully is the product of the fidelities of each SWAP operation in π_s. This probability is

$$c(\pi_s) = \prod_{h=0}^{|\pi_s|-1} \delta\left(\pi_{s,h_0}, \pi_{s,h_1}\right) \tag{3.10}$$

Now, given a solution $s = (\lambda_0, \lambda_1, \ldots, \lambda_{L-1})$ to the QLP and random seed r, let $\pi_1 \ldots \pi_{L-1}$ be sequences of SWAP operations to permute each λ_i to λ_{i+1}. Then, the probability $f_s(s,r)$ that all SWAP operations execute successfully is the product of $c(\pi_1) \cdot c(\pi_2) \cdot \ldots \cdot c(\pi_{L-1})$ because all SWAP operations of each SWAP sequence are treated as independent events, and all events must be successful for the event that all SWAP operations execute without error to occur. This probability is

$$MAXIMIZE\ f_s(s,r) = \prod_{\lambda_{i-1},\lambda_i \in s} c\left(w(\lambda_{i-1}, \lambda_i, r)\right) \tag{3.11}$$

where s is a partial solution to the QLP and r is a random seed.

Single Objective Function: Since the codomain of both objective functions is a probability of successfully executing operations, objective functions 1 and 2 can be combined into a single objective function.. Moreover, the events of each objective function are independent. The single objective function for the QLP is

$$MAXIMIZE\ f_{obj}(s,r) = f_q(s) \cdot f_s(s,r) \tag{3.12}$$

where s is a partial solution to the QLP and r is a random seed.

The value of f_{obj} for a given s is the probability that a given circuit transpiled to a given backend with layouts determined by the decision variables of s and SWAP layers generated using r will execute successfully. For the circuit to execute successfully, all 1- and 2-qubit gate operations must execute correctly as well as all SWAP operations required to permute the logical

qubits in $\pi_1 \ldots \pi_{L-1}$. Since f_q captures the probability of the former and f_s the probability of the latter, $f_q \cdot f_s$ is the probability both events occur.

Note that f_{obj} only deals with gate-related fidelities. In reality, numerous other sources of error degrade the fidelity further (such as T_1 and T_2 related errors). These errors are impossible to accurately account for due to not knowing the state of the wave-function at an arbitrary point in the circuit. Nonetheless, T_1 and T_2 related errors are mitigated by producing circuits of minimum depth. Since minimizing gate related errors generally results in circuits of minimum depth, T_1 and T_2 errors are subsequently minimized via f_{obj}.

3.3.1.5 Surrogate Objective Functions

In the literature, an approximation of an objective function is commonly called a *surrogate objective function* or *surrogate fitness function* (e.g. Brownlee, Woodward, and Swan [37]). The execution time bottleneck of f_{obj} is the TokSP approximation algorithm. To mitigate this bottleneck, the upper- and lower-bound number of SWAPs required to permute between each λ_{i-1} and λ_i are used as the basis for a surrogate objective function.

Let D_i be the sum of all distances of logical qubits from their current vertices in λ_{i-1} to their target vertices in λ_i. Then Miltzow, et al. state that their algorithm needs at most $4 \cdot D_i$ SWAPs and at least $\frac{D_i}{2}$ SWAPs, to permute between λ_{i-1} and λ_i. Since each SWAP operation decomposes to 3 C_{NOT} operations, the upper bound number of C_{NOT} gates required is $\eta_u = 3 \cdot 4 \cdot D_i$, and the lower bound is $\eta_l = 3 \cdot \frac{D_i}{2}$. Let ϖ_{min} be the minimum 2-qubit success rate and ϖ_{max} the maximum 2-qubit success rate. Then $\varpi_{min}^{\eta_u}$ bounds the fidelity of performing all η_u C_{NOT} operations from below, and $\varpi_{max}^{\eta_l}$ bounds the fidelity of performing all η_l C_{NOT} operations from above. The functions below define how to calculate the upper and lower bound fidelities of performing all SWAP sequences $\pi_1 \ldots \pi_{L-1}$.

Let $d_1(\lambda_i, \lambda_j, q_k)$ be the length shortest unweighted path from $P_l = \lambda_i^{-1}(q_k)$ to $P_m = \lambda_j^{-1}(q_k)$ in G. Given function d_1, D_i is:

$$D_i = \sum_{j=0}^{M-1} d_1(\lambda_{i-1}, \lambda_i, q_j) \tag{3.13}$$

For the QLP, there are $L-1$ SWAP sequences $\pi_1, \ldots, \pi_{L-1}$. Each SWAP sequence has an associated D_i since $\pi_i = w(\lambda_{i-1}, \lambda_i)$. Let $\eta_u(D_i) = 3 \cdot 4 \cdot D_i$ and $\eta_l(D_i) = 3 \cdot \frac{D_i}{2}$. Given $D_1 \ldots D_{L-1}$, the lower- and upper-bound fidelities of executing all SWAP sequences are then:

$$f_{s,l}(s) = \varpi_{min}^{\varrho_u} \tag{3.14}$$

s.t.

$$\varrho_u = \sum_{i=1}^{L-1} \eta_u(D_i)$$

$$f_{s,u}(s) = \varpi_{max}^{\varrho_l} \tag{3.15}$$

s.t.

$$\varrho_l = \sum_{i=1}^{L-1} \eta_l(D_i)$$

Equation 3.14 defines the lower-bound and Equation 3.15 the upper-bound. In $f_{s,l}$, ϱ_u is the upper-bound number of C_{NOT} operations that need to be performed for all π_i. In $f_{s,u}$, ϱ_l is the lower-bound number of C_{NOT} operations that need to be performed for all π_i. Given that ϖ_{min} is the minimum 2-qubit success rate of some edge $e_{ij} \in E(G)$, in the worst-case all ϱ_u C_{NOT} operations occur on e_{ij}; thus, $\varpi_{min}^{\varrho_u}$ is the lower-bound fidelity. Via the same logic, given that ϖ_{max} is the maximum 2-qubit success rate of some edge $e_{ij} \in E(G)$, in the best-case ϱ_l C_{NOT}

operations occur on e_{ij}; thus, $\varpi_{max}^{\varrho_l}$ is the upper-bound fidelity. Now, the upper and lower bound objective functions are defined as follows:

$$MAXIMIZE\ f_{obj,l}(s) = f_q(s) \cdot f_{s,l}(s) \tag{3.16}$$

$$MAXIMIZE\ f_{obj,u}(s) = f_q(s) \cdot f_{s,u}(s) \tag{3.17}$$

Here, $f_{s,l}$ ($f_{s,u}$) is a lower (upper) bound on the probability that all SWAP sequences execute successfully. $f_{obj,l}$ and $f_{obj,u}$ bound the true objective function f_{obj} by considering lower- and upper- bounds developed by Miltzow et al. By construction, for any partial solution $s = (\lambda_0, \lambda_1, \ldots \lambda_{L-1})$ to the QLP, it will always be the case that:

$$f_{obj,l}(s) \leq f_{obj}(s) \leq f_{obj,u}(s) \tag{3.18}$$

The functions $f_{obj,u}$ and $f_{obj,l}$ are the candidate surrogate functions considered. The former is chosen rather than the latter for two reasons. First, due to the underestimated fidelity of swapping (consequence of Equation 22) in $f_{obj,l}$, maximizing $f_{obj,l}$ results in solutions that tend to use very few (if any) SWAPs. This is unlikely to be advantageous; while swapping is costly, performing some SWAPs is generally beneficial. That is, performing a small number of SWAPs can result in better layouts and the benefit of using a better layout can outweigh the cost of doing a few SWAPs. Second, the link(s) between one or more pairs of physical qubits could be dead (i.e. have a fidelity of 0). If $f_{obj,l}$ were used in such a problem instance, the surrogate objective score would be uniformly 0.

3.3.2 Complexity and Difficulty

The solution space of the QLP consists of all sequences of P2L assignments and all sequences of SWAP operations to permute between layouts. The solution space is infinite in size because the set of SWAP sequences to permute between a given λ_i and λ_j is infinite. Due to the infinite

size of the solution space, a design choice is made in this research effort to only consider a finite subset of the solution space.

In Section 3.3.1, a solution to the QLP is represented as $s = (\lambda_0, \lambda_1, \ldots, \lambda_{L-1})$. In this representation, SWAP sequences are considered to be determined by P2L mappings and the choice of a random seed. That is, given a λ_i, λ_j, and random seed r, the approximation function $w(\lambda_i, \lambda_j, r)$ deterministically yields a single SWAP sequence to permute λ_i to λ_j. Since the TokSP solver only yields a single SWAP sequence and the solution space consists of all SWAP sequences to permute between λ_i and λ_j, the search space[1] does not represent all solutions in the solution space. Nevertheless, many of the potential SWAP sequences are obviously bad (e.g. those that include a 2-cycle $\pi_s = (\ldots, e_{ij}, e_{ij}, \ldots)$). Moreover, since the optimality gap is bounded in the TokSP solver, this research acknowledges that given some inputs, the TokSP solver yields optimal or near-optimal sequences of SWAPs, and given other inputs, sequences of SWAPs no worse than $4 \cdot OPT$.

Since each permutation consists of N elements, and there are L layers, the search space is of size $O((N!)^L)$. Thus, the search space is finite. For the QLP, the decision variables are discrete-valued (when considering the canonical representation). Thus, the QLP satisfies Talbi's definition of a combinatorial optimization problem [6].

The QLP is composed of two sets of sub-problems. The first set of sub-problems is a sequence of L assignment problems. For each layer of the circuit, the goal is to find an assignment of P2L assignments that maximizes the layer's probability of executing successfully. The second set of sub-problems is a sequence of $L - 1$ instances of the TokSP. The role of the TokSP in the QLP is to find sequences of SWAP operations of maximal success probability to permute between each

[1] Following Talbi [5], a solution space is the set of all conceivable solutions to a problem. A search space is induced by a representation and is the set of all solutions represented by that representation.

layout λ_{i-1} and λ_i. Thus, the QLP is a combination of assignment problems and TokSPs. The optimal solution to the QLP is a balance between high-fidelity P2L assignments for each λ_i and high-fidelity sequences of SWAP operations to permute between each λ_{i-1} and λ_i.

Finding optimal solutions that balance high-fidelity P2L assignments for each λ_i and high-fidelity sequences of SWAP operations to permute between each λ_{i-1} and λ_i is a computationally difficult task. Typically, optimal P2L mappings for each layout result in low-fidelity SWAP sequences. Conversely, high-fidelity SWAP sequences generally result in low-fidelity P2L mappings for each λ_i. In fact, Tan and Cong [17] prove that the decision version of the QLP is in NPC.

The NP-hardness alone of the QLP does not justify the use of metaheuristics. Even if a problem is NP-Hard, small instances can be solved via exact approaches. For the QLP, this research seeks to find good approximate solutions to problem-instances that cannot be solved via exact approaches and that push NISQ-era QCs beyond the boundary of computations that can be performed on classical computers.

Beyond problem-instance size, Talbi states that the structure of instances is also a factor in assessing the applicability of metaheuristics to a problem. Indeed, for the QLP there exist trivially solvable instances (e.g. in the presence of uniform qubit and link reliabilities, a circuit with no C_{NOT} operations or on a complete graph topology). However, this research focuses on problem-instances of the QLP with challenging structures, such as circuits with a hub-and-spoke topology (following Kamaka [19]) transpiled to QCs for which the topologies are general graphs with limited connectivity. The power of QCs is largely due to their capability to entangle qubits (as well as QCs ability to place qubits in superposition), which on IBM's QCs and similar architectures can only be achieved via C_{NOT} operations.

Due to the complexity, difficulty, and combinatorial nature of the QLP, as well as the instance-sizes and structures of the QLP this research seeks to solve, there are good reasons to believe that metaheuristics could be a fruitful approach to finding good approximate solutions to the QLP.

3.3.3 Requirements Analysis

This section outlines the requirements of a QLP-solver to compete with SOTA quantum program transpilers. Talbi provides three criteria that must be analyzed for a metaheuristic [6]. The first, search time requirements, relates to efficiency and analyzes how much time is available for the optimization algorithm to find quality solutions. The second, quality of solutions requirements, relates to effectiveness and analyzes the sense in which the solutions obtained by the solver must be near-optimal.

Search Time Requirements: For the QLP in the context of the IBMQ, the execution time of a proposed QLP-solver should be significantly less than the period with which IBM updates their calibration data because once the calibration data is updated, the reported 1- and 2-qubit error rates can be very different. Since qubit and link error rates are key parameters in the objective functions, if they change, so does the objective space. Next, the QLP-solvers proposed in this research are anticipated to run much more slowly than SOTA QLP-solvers. The proposed QLP-solvers are intended to spend more time searching for high-quality solutions via metaheuristics to find higher quality local optima.

Quality of Solutions Requirements: Solutions obtained from the QLP-solvers proposed in this research should yield transpiled circuits that return the correct state-vector more often than the circuits generated by SOTA QLP-solvers.

3.3.4 Design of Metaheuristic Algorithms

This section outlines the design choices that must be made in the development of meta-based QLP-solvers. This research effort seeks to develop both single-solution-based and population-based metaheuristic solvers for the QLP. The single-solution-based solvers focus on intensification of a single solution to settle in on a locally optimal solution. The population-based solvers focus on diversification via a population of candidate solutions to evolve prominent features of strong solutions in the search space.

First, representations for the single-solution based and population-based metaheuristic algorithms are defined. Then, Sections 3.3.4.1 through 3.3.4.4 provide candidate meta-based QLP-solvers.

Representation for Single-Solution-Based Solver: The proposed single-solution-based solvers work directly on the phenotypic representation. That is, given a partial solution s to the QLP, each $\lambda_i \in s$ is a permutation of the trivial P2L mapping defined in Section 3.3.1.1. Thus, the phenotypic representation works on the partial solution $s = (\lambda_0, \lambda_1, \dots, \lambda_{L-1})$ defined in Section 3.3.1.4. Below, the completeness, connexity, and efficiency of the phenotypic representation are addressed.

- Completeness: As discussed in Section 3.3.2, all solutions to the QLP are by design not represented by the partial solution s.
- Connexity: Via an n-exchange search operator operating on each $\lambda_i \in s$, all solutions associated with the problem are reachable from each other. This is because via a 1-exchange search operator operating on each λ_i, all permutations of λ_i are generated (a proof is offered by Conrad [38]). Moreover, any n-exchange operation for $n > 1$ also includes all 1-exchanges.

- Efficiency: Applying n-exchange operations to a given $\lambda_i \in s$ is computationally efficient ($O(n)$).

Representation for Population-Based Solver: Computationally efficient evolutionary operators (e.g. mutation and crossover) acting on the phenotypic representation would produce infeasible solutions. For instance, consider a mutation operator that randomly alters an element $P_j \mapsto q_k$ of a P2L mapping by reassigning $P_j \to q_l : k \neq l$. In this case, the mutated P2L mapping is invalid because a P2L mapping must be bijective and some other $P_m \mapsto q_l$. Thus, to ensure mutation and crossover operators always yield feasible solutions, the population-based solvers encode solutions via the random-key encoding discussed in Section 3.3.1.2. While the phenotypic representation uses partial solution $s = (\lambda_0, \lambda_1, \dots, \lambda_{L-1})$, this genotypic representation uses partial solution $x = (\chi_0, \chi_1, \dots, \chi_{L-1})$. Since a given χ_i decodes as a λ_i, the completeness argument for the genotypic representation is the same as for the phenotypic representation since every assignment of a λ_i is represented by some assignment of χ_i. Talbi provides justification of the connexity and efficiency of random-key encoding.

3.3.4.1 Variable Neighborhood Descent QLP-Solver

In this section, two VND-based QLP-solvers are proposed. As discussed in Section 2.5.1.1, the VND is classified as a single-solution based metaheuristic. The design of both VND-based QLP-solvers begins with definition of an initial solution, neighborhoods, and an order of their application [6].

- Initial solution: The initial solution is generated either by randomly sampling the search space, or by providing a solution obtained by another metaheuristic (e.g. the best solution found by the BRKGA-based QLP-solver).
- VND requires the definition of a set of neighborhood structures $N_1, \dots, N_l$ for l distinct neighborhoods. To minimize overall execution time, the cardinality of each resulting

neighborhood must be considered. Table 13 define the neighborhoods studied in this research effort.

Variable/Function Name	Description
G	Graph representing the topology of target architecture, as defined in Section 3.3.1.1.
N	Number of physical qubits; $N = \|V(G)\|$.
L	Number of layers in the circuit.
$\hat{E}(x)$	The undirected edge-set of graph x.
E	$E = \left\|\hat{E}(G)\right\|$.
κ_N	The complete graph sharing the vertex-set of G.
$P(x)$	The powerset of set x.
$\xi(x)$	The set containing exactly the elements of the multiset x.
$\nu_p(l)$	The number of occurrences of element l in multiset p.
$V(x)$	The vertex-set of graph x.
s	A partial solution $s = (\lambda_0, \lambda_1, \ldots, \lambda_{L-1})$ to the QLP.
N_i	Neighborhood name, not related the number of physical qubits N (above).

Table 12: Notation used in Table 13: Neighborhood functions devised for VND algorithm.

Neighborhood Name	Description	Size
N_1	All 1-exchanges over $\hat{E}(G)$ of each $\lambda_i \in s$.	$L \cdot E$
N_2	All 1-exchanges over $\hat{E}(\kappa_N)$ of each $\lambda_i \in s$.	$L \cdot \frac{N(N-1)}{2}$
N_3	All 2-exchanges over $\hat{E}(G)$ of each $\lambda_i \in s$.	$L \cdot E \cdot (E-1)$
N_4	All 2-exchanges over $\hat{E}(\kappa_N)$ of each $\lambda_i \in s$.	$L \cdot \left(\frac{N(N-1)}{2}\right)^2$
N_5	$\forall \lambda_i, \lambda_{i+1} \in s$, all (1-exchanges over $\hat{E}(\kappa_N)$ of λ_i) $\times$ (1-exchanges over $\hat{E}(\kappa_N)$ of λ_{i+1}).	$(L-1) \cdot \left(\frac{N(N-1)}{2}\right)^2$
N_6	$\forall \lambda_i, \lambda_{i+1} \in s$, all (1-exchanges over $\hat{E}(G)$ of λ_i) $\times$ (1-exchanges over $\hat{E}(G)$ of λ_{i+1}).	$(L-1) \cdot E^2$
N_7	*Math*: *Let* $A = \{\lambda_i \in \xi(s): \nu_s(\lambda_i) > 1\}$. $\forall \lambda_i \in A$, all 1-exchanges over $\hat{E}(\kappa_N)$ of λ_i $\forall \lambda_j \in s : \lambda_j = \lambda_i$. *English*: Here, A is a set that holds all λ_i that occur more than once in s. For all layouts $\lambda_i, \lambda_j, \ldots, \lambda_k$ that are the same in s, create all 1-exchanges of λ_i and duplicate across $\lambda_j \ldots \lambda_k$.	$L_{repeats} \cdot \frac{N(N-1)}{2}$, where $L_{distinct}$ is the number of unique layouts that occur more than once in s.

N_8	For all layouts that have (q_i, q_j) placed in (P_k, P_l), exchange P_k with P_m and P_l with P_n in each layout such that $P_m \neq P_n \wedge q_i \neq q_j$.	$C_{distinct} \cdot \left(\frac{N(N-1)}{2}\right)^2$, where $C_{distinct}$ is the number of unique logical qubit pairs (q_i, q_j) that are placed in physical qubit pair (P_k, P_l) in more than one layout of s.
N_9	For all layouts that have q_i placed in P_j, exchange P_j with P_k such that $P_j \neq P_k$.	$C^*_{distinct} \cdot \frac{N(N-1)}{2}$, where $C^*_{distinct}$ is the number of unique logical qubits q_i that are placed in physical qubit P_j in more than one layout of s.

Table 13: Neighborhood functions devised for VND algorithm.

- Finally, the VND algorithm requires an objective function to optimize. For the VND QLP-solver, the objective functions to optimize are f_{obj} and $f_{obj,u}$.

The neighborhoods in Table 13 are not defined arbitrarily. In the design of the neighborhoods, the goal is to transform solutions into better solutions. For the QLP, the general structure of a good solution $M_h = (\lambda_0, \pi_1, \ldots, \pi_{L-1}, \lambda_{L-1})$ is one which the $\sum_i |\pi_i|$ is minimal and $f_q\big((\lambda_0, \lambda_1, \ldots, \lambda_{L-1})\big)$ is maximal. In other words, good solutions contain π_is of minimal aggregate cardinality and λ_js that maximize the probability that all 1- and 2-qubit operations succeed. In the following three paragraphs, consider the neighborhoods are visited in ascending order (based on neighborhood subscript) by a VND algorithm.

N_1 through N_4 use 1- and 2-exchange operators acting on each $\lambda_i \in s$ to achieve a solution $s' = (\lambda'_0, \lambda'_1, \ldots, \lambda'_{L-1})$ such that no defined 1- or 2-exchange operator acting on any $\lambda'_i \in s'$ increases the fitness score of s'. In other words, they achieve a locally optimal solution s' in which small perturbations to each λ'_i do not yield a solution s'' of higher fitness.

After N_1 through N_4 achieve solution s', neighborhoods N_5 and N_6 use pairs of 1-exchange operators, each acting on one of the adjacent layouts $(\lambda'_i, \lambda'_{i+1}) \in s'$ with the goal of reducing the aggregate distance between all pairs of consecutive layouts in s'. For instance, a 1-exchange operator acting on a given λ'_i may not yield a solution s'' of higher fitness than s'. However, a pair of 1-exchange operators, one acting on λ'_i and the other on λ'_{i+1}, may yield a solution s'' of higher fitness than s'. This situation is anticipated because simultaneous exchanges in two adjacent layouts enable perturbations to s' that reduce the distance between λ'_i and λ'_{i+1}.

At the point N_7 is reached, it is anticipated that many of the layouts in the solution s'' are identical because N_5 and N_6 are anticipated to significantly reduce the distance between each pair of consecutive layouts. When the distance between two consecutive layouts is small, the P2L mappings are nearly identical. N_7, N_8, and N_9 exploit the similarity of consecutive layouts via 1- and 2-exchange operators acting on multiple (similar) layouts in s''. Simultaneous perturbation of multiple similar layouts is anticipated to yield increases in fitness to s'' that cannot be achieved using the previous neighborhoods. This is because $N_7, N_8,$ and N_9 create larger perturbations to s'' than previous neighborhoods and these perturbations are anticipated to permute frequently used logical qubits to stronger subgraphs of G (i.e. subgraphs of G with lower error rates). For instance, consider a frequently entangled logical qubit pair (q_i, q_j). Suppose that in multiple layouts of s'' logical qubit q_i is placed in P_k and q_j in P_l. Now instead suppose q_i is placed in P_m and q_j in P_n and the success rate of link $e_{kl} = (P_k, P_l)$ is much less than that of link $e_{mn} = (P_m, P_n)$. Movement of q_i to P_m and q_j to P_n increases circuit fidelity by up to $\left(\frac{e_{mn}}{e_{kl}}\right)^g$ times if (q_i, q_j) entangle g times.

Now, the first VND-based QLP-solver algorithm is presented. Due to the poor asymptotic execution time of f_{obj} and large number of anticipated objective function calls by the VND, a surrogate objective function $f_{obj,u}$ is used conditionally via a Boolean variable γ (see Appendix

C for time-complexity analysis of the objective functions). Given that the worst-case time complexity of f_{obj} is $O(LN^3)$ while that of $f_{obj,u}$ is $O(LN^2)$, significant runtime performance gains are achievable by using the surrogate objective function.

For the first proposed VND-based QLP-solver, if $\gamma = \text{True}$, then the surrogate objective function is first used to produce a high-quality solution (albeit, likely of overestimated quality since the upper-bound surrogate uses the lower-bound number of SWAP operations that must be performed and the highest-fidelity link success rate). Nonetheless, even in this approximation, the aggregate distance ϱ_l between all λ_{i-1} and λ_i is anticipated to be reduced considerably since the surrogate objective function yields low-fidelity objective scores when ϱ_l is large. For instance, consider a plausible maximum 2-qubit link success rate of $\varpi_{max} = 0.99$. For a sufficiently large ϱ_l, $f_{obj,u} = \varpi_{max}^{\varrho_l}$ approaches 0. The first VND-based QLP-solver is provided in Algorithm 18. Note that this algorithm is really two distinct QLP-solvers (when $\gamma = \text{True}$ and when $\gamma = \text{False}$).

Algorithm VND-based QLP-solver.

Inputs: An initial solution s_0 (if available) and a Boolean γ denoting whether to use the surrogate objective function prior to the true objective function.

$s = s_0$ or $generate\_random\_solution()$;
$X = (N_1, N_2, N_3, N_6, N_7, N_9, N_8)$; /* Sequence of neighborhoods to explore */
$l_{max} = |X|$; /* l_{max} is the number of elements in X */

If $\gamma = True$ **Then**
  /* First, run VND algorithm with surrogate objective function */
  $l = 0$;
  While $l < l_{max}$ **Do**
    $Find$ the best neighbor s' of s in $X_l(s)$ via $f_{obj,u}$;
    If $f_{obj,u}(s') > f_{obj,u}(s)$ **Then**
      $s = s'$; $l = 0$;
    Otherwise
      $l = l + 1$;
  End While

/* At this point, if $\gamma = True$, s holds the best solution found from the VND running with the Surrogate objective function */

/* Run VND algorithm with true objective function */
$l = 0$;
While $l < l_{max}$ **Do**
  $Find$ the best neighbor s' of s in $X_l(s)$ via f_{obj} ;
  If $f_{obj}(s') > f_{obj}(s)$ **Then**
    $s = s'$; $l = 0$;
  Otherwise
    $l = l + 1$;
End While

Output The best solution found s, which is locally optimal with respect to all neighborhoods.

Algorithm 18: VND-based QLP-solver. Neighborhoods N_4 and N_5 are omitted due to their large cardinalities.

The second VND-based QLP-solver is presented to address the computational expense of the objective function f_{obj}. This variant, called the Gradient-VND QLP-solver, uses the surrogate $f_{obj,u}$ to evaluate candidates at each iteration. Then, the top k candidates are scored via the actual objective function, f_{obj}. In doing so, f_{obj} is presented with a limited number of candidates that are likely to increase the current best score obtained from f_{obj}. This variant of the VND was devised with influence from Brownlee et al. [37] as well as Ky et al. [39], who discuss the

applicability of surrogate fitness functions to poor runtime (costly) objective functions. The Gradient-based QLP-solver is provided in Algorithm 19.

Algorithm Gradient-VND-based QLP-solver.

Inputs: An initial solution s_0 (if available) and a maximum number of evaluations k to perform with f_{obj} per iteration of the VND.

$s = s_0$ or $generate\_random\_solution()$;
$X = [N_1, N_2, N_3, N_6, N_7, N_9, N_8]$; /* Sequence of neighborhoods to explore */
$l_{max} = |X|$; /* l_{max} is the number of elements in X */

/* First, run VND with surrogate objective function */
$l = 0$;
$x' = s$; /* persist best found solution in f_{obj} objective space */

While $l < l_{max}$ **Do**
  $K = find$ the best k neighbors $s'_1, \dots, s'_k$ of s where $f_{obj,u}(s'_i) > f_{obj,u}(s)$;
  If $|K| = 0$ **Then**
    $l = l + 1$; /* No improvements found */
  Otherwise
    $improvement = \emptyset$;
    For z in K **Do**
      If $f_{obj}(z) > f_{obj}(x')$ **Then** $x' = z$; $improvement = z$;
    End For

    If $improvement \neq \emptyset$ **Then**
      $s = improvement$; $l = 0$;
    Otherwise
      $l = l + 1$;
End While

Output Best solution found x', which is locally optimal with respect to all neighborhoods.

Algorithm 19: Gradient VND-based QLP-solver.

3.3.4.2 Genetic Algorithm QLP-Solver

The GA-based QLP-solver uses a BRKGA as described in Section 2.5.2.1. Reference Algorithm 11 of Section 2.5.2.1 for a full description of the BRKGA. A BRKGA requires an initial population, mutant population generator, crossover operator, and a selection mechanism. The GA-based QLP-solver uses the following operators:

- Initial population: The initial population is generated by randomly sampling the search space. As each decision variable takes on a value from $[0,1]$ and there are $L \cdot N$ decision variables, random individuals (solutions) are generated by uniformly sampling $[0,1]^{L \cdot N}$.
- Mutation: BRKGA's do not have a mutation operator that acts on individuals. Rather, the population P is augmented with a set P_m of mutants. Each $s_m \in P_m$ is randomly generated using the same mechanism as the individuals in the initial population.
- Crossover: BRKGA's typically use parameterized uniform crossover. In BRKGA's, the population is partitioned into an elite population P_e and a non-elite population $P_{\bar{e}}$ at the beginning of each generation of an evolution. The elite partition consists of the $e = \lfloor p \cdot p_e \rfloor$ highest-fitness members of the population for a population size p and elite population portion p_e and the non-elite partition consist of the remaining members of the population. Then, parent p_1 is selected from P_e and parent p_2 is selected from $P_{\bar{e}}$. Prior to running the GA, a hyperparameter ρ_a is assigned a real-value from $(0.5,1]$. Finally, crossover is performed by iterating over all loci of p_1 and p_2, and for each locus selecting the allele from p_1 with probability ρ_a and the allele from p_2 with probability $1 - \rho_a$.
- Selection: Each iteration of the BRKGA updates the population as follows: $P = P_m \cup P_e \cup P_c$. Here, P_c is the offspring generated via crossover.

Thus, the BRKGA manages diversification via the introduction of mutants into the population and intensification via the parameterized uniform crossover. The BRKGA also requires the following:

- An objective function to optimize. Due to the large number of function evaluations performed by the BRKGA, the surrogate objective function $f_{obj,u}$ is used. Details about why this surrogate was selected are in Section 3.3.1.5.

- The number of decision variables. For the QLP under the random-key encoding, there are $L \cdot N$ decision variables. See Section 3.3.1.2 for more information.

3.3.4.3 Evolution Strategies QLP-Solver

The evolution strategies-based QLP-solver primarily uses MOES for stochastic population-based search and Nelder-Mead for local search at the end of each evolution. Section 2.5.2.2 presents MOES. Thus, the ES-based solver is a hybrid QLP-solver (as discussed in Section 3.3.4.4). For both MOES and Nelder-Mead, the true objective function f_{obj} is used.

3.3.4.4 Hybrid QLP-Solvers

A hybrid QLP-solver is composed of both a population-based and single-solution based QLP-solver p_{meta} and s_{meta}, respectively, and denoted p_{meta}+s_{meta}-based QLP-solver. First, p_{meta} is used to find a solution or set of solutions to the QLP. Next, the best solution(s) found from p_{meta} are used as input to s_{meta} as an initial solution(s). If s_{meta} is the VND, the best solutions(s) found from p_{meta}+VND are guaranteed to be locally optimal with respect to the defined neighborhood structures of the VND.

3.3.5 Parameter Tuning

Each metaheuristic algorithm defined in section 3.3.4 has hyperparameters that must be set. In this section, the hyperparameters for the VND, BRKGA, and ES solvers are enumerated, and the configurations of the algorithms are discussed.

VND Parameters: Three hyperparameter choices must be made for the VND. The first selects the neighborhood functions to descend through. The second defines the order in which to descend through the selected neighborhoods. The third determines whether to accept the first improving neighbor in a neighborhood or to select the best neighbor in a neighborhood. The latter choice requires the complete evaluation of the neighborhood, which is computationally expensive for large-sized neighborhoods.

- Selected Neighborhoods: In Section 3.3.4.1, nine neighborhood functions are defined. Of the nine neighborhood functions, the following are used in the VND algorithm for this research: $N_1, N_2, N_3, N_6, N_7, N_8, N_9$. Neighborhoods N_4 and N_5 are omitted due to their larger cardinality (space complexity $O(N^4)$).
- Order of Neighborhoods: The following order of neighborhoods is used for the VND algorithm in this research: $N_1, N_2, N_3, N_6, N_7, N_9, N_8$. This order is chosen based on the sizes of the neighborhoods and the maximum number of moves (i.e. transposition operations) each neighborhood function N_l applies to a solution s. In Table 14, the select neighborhood functions are first ranked from smallest to largest by maximum move count. When maximum move count is the same between neighborhoods functions, the space complexity is the tie-breaker.

Neighborhood	Space Complexity	Perturbation Performed	Maximum Move Count
N_1	$O(LE)$	1-exchange over the $E(G)$ of layout $\lambda_i \in s$.	1
N_2	$O(LN^2)$	1-exchange over $E(\kappa_N)$ of layout $\lambda_i \in s$.	1
N_3	$O(LE^2)$	2-exchange over $E(G)$ of layout $\lambda_i \in s$.	2
N_4	$O(LN^2)$	2-exchange over $E(\kappa_N)$ of layout $\lambda_i \in s$.	2
N_6	$O(LE^2)$	1-exchange over $E(G)$ of layout $\lambda_i \in s$ and another 1-exhange over $E(G)$ of layout $\lambda_{i+1} \in s$.	2
N_7	$O(LN^2)$	For all layouts $\lambda_i, \lambda_j, \dots, \lambda_k \in s$ where $\lambda_i = \lambda_j = \cdots = \lambda_k$, 1-exchange of λ_i over $E(\kappa_N)$ duplicated across $\lambda_j \dots \lambda_k$.	L
N_9	$O(LN^2)$	For all layouts $\lambda_i, \dots, \lambda_k \in s$ that have q_i placed in P_j, exchange P_j with P_k in each layout.	L
N_8	$O(LN^4)$	For all layouts $\lambda_i, \dots, \lambda_k \in s$ that have (q_i, q_j) placed in (P_k, P_l), exchange P_k with P_m and P_l with P_n in each layout.	$2L$

Table 14: Ranking select neighborhoods by maximum move count and space complexity.

- Acceptance Criterion: Due to the low-order polynomial size of each neighborhood, all neighborhoods are completely evaluated to select the best neighbor.

BRKGA Parameters: Section 2.5.2.1 presents the hyperparameters of a BRKGA. In Table 15, all BRKGA hyperparameters are enumerated and configured.

Parameter Name	Parameter Value
Population Size (p)	$N \cdot L$
Elite Population Portion (p_e)	$p \cdot 0.15$
Mutant Population Portion (p_m)	$p \cdot 0.10$
Bias (ρ_a)	0.65
Termination Criterion: Maximum Number of Generations (G)	5000

Table 15: Parameters and selected values for BRKGA-based QLP-solver.

As discussed in Section 2.5.2.1, Goncalves and Resende [24] recommend a population size p of R, where R is the number of decision variables. For the QLP under random-key encoding, $R = L \cdot N$. Accordingly, a population size of $L \cdot N$ is selected. The population size scales linearly with input size of the problem-instance, where the input size is defined by the number of decision variables $L \cdot N$. In effect, as input size increases, the number of candidate solutions evaluated per generation of the BRKGA scales linearly. This is desired because the solution-space size of the QLP also scales with L and N. Overall, the BRKGA explores at most $\frac{G \cdot L \cdot N \cdot 100}{(N!)^L}$% of the solution-space.

Next, as also discussed in Section 2.5.2.1, Goncalves and Resende recommend the elite partition p_e be between 10% and 25% of p and the mutant partition p_m be between 5% and 20% of p. In addition, they recommend the bias ρ_a be greater than 50%. In this research effort, p_e, p_m, and ρ_a are jointly configured considering of the recommendations of Goncalves and Resende, and considering intensification and diversification. p_e and ρ_a manage intensification while p_m manages diversification. That is, higher values of p_e and ρ_a increase the probabilities that elite members of the population survive to the next generation and offspring are generated

with genes from elite members, respectively. Higher values of p_m result in an evolving population consisting of more random solutions, thereby diversifying the evolving population. For the QLP, p_e is set to 15% of p, p_m to 10% of p, and ρ_a to 65%. Intensification is slightly favored (given that $p_e > p_m$ and $\rho_a > 0.5$) for the QLP due to the large size of the solution-space. Experimentation indicated higher intensification yields faster convergence on lower-quality solutions and lower intensification yields slower convergence (if any). At the extreme, low intensification and high diversification mirrors random search for the QLP.

Finally, experimental results show that with the selected configuration of all other hyperparameters of the BRKGA, signs of convergence by the algorithm are apparent within 5000 generations for most test cases. Convergence in this research effort is considered "apparent" when the fitness of the best member of the evolving population does not increase for $\frac{G}{2}$ generations for some $G \gg 10$.

ES Parameters: The ES-based QLP-solver's hyperparameters are tuned as shown in Table 16.

Parameter Name	Selected Parameter Value	Description
Problem_Type	2	0 for multiple objective, 1 for constrained single objective, and 2 for unconstrained single objective.
Num_Evolutions	10	Number of evolutions to perform.
Num_Generations	5000	Maximum number of generations in a given evolution.
p	$10 \cdot N \cdot L$	Evolving population size.
μ	$\frac{1}{5} \cdot p$	Number of parents in crossover.
λ	$\frac{4}{5} \cdot p$	Number of children in crossover.
κ	$M \cdot L$	Maximum lifetime, in generations in crossover.
ρ	2	Number of parents used in crossover.

Selection	$truncation$	Selection mechanism employed.
Recombination	$uniform$	Type of crossover to perform.
Use_Amoeba	1	Apply Downhill Simplex algorithm after each evolution (1), to best solution obtained after all evolutions (2), do not apply (0).

Table 16: Parameters and selected values for the ES-based QLP-solver. Adapted from Lill and Smith [23].

First, since the QLP is modeled as a single-objective unconstrained optimization problem, the problem type is chosen to be 2 (MOES considers the QLP as unconstrained because it does not consider bounding constraints as constraints). Second, because MOES is a stochastic algorithm, 10 evolutions are run, which is sufficient to allow for statistical analysis while still being computationally feasible within the time available for the high-performance computers (HPC) used to transpile the test cases. Third, experimentation with MOES showed that 5000 generations is sufficient to obtain indications of convergence (i.e. the average standard deviation of strategy parameters approach zero) for select problem-instances.

Fourth, Lill recommends that the overall evolving population size p be an order of magnitude greater than the number of decision variables; an empirically derived rule he and colleagues have established over years of applying MOES to a variety of problems. For a given QLP problem-instance, there are $L \cdot N$ decision variables. Accordingly, p is set to $10 \cdot L \cdot N$. Fifth, Beyer and Schwefel suggest that the number of children in the population λ be four to five times greater than the number of parents μ [40]. As such, λ is configured to be $\frac{4}{5}$ of the evolving population and μ is configured to be $\frac{1}{5}$ of the evolving population. As with the BRKGA, the size of the evolving population scales with input size. After configuration of λ and μ, κ was initially set to 50 generations. However, this lifespan was too long for many of the easier test cases and not long enough for the more challenging test cases. Experimental results found that a κ of about $M \cdot$

L provided long enough lifespan to yield sufficient perturbations to members of the evolving population for most problem-instances.

Sixth, a truncation selection (as opposed to tournament selection scheme) is used since truncation is the standard selection mechanism employed traditionally in ES. Seventh, uniform crossover is chosen based on experimentation exhibiting that this type of crossover yields the highest-quality results for the QLP (when compared to mutation-only and diagonal crossover). Finally, the Nelder-Mead downhill simplex method is applied after each evolution and attempts to further improve the fitness score of the best solution found in the evolution.

3.3.6 Devised QLP-Solvers

In this research effort, numerous candidate QLP-solvers are proposed in Sections 3.3.4.1 through 3.3.4.3. In addition, Section 3.3.5 provides parameter tuning for all metaheuristic algorithms employed in each candidate QLP-solver. As defined in section 3.3.4.4, combinations of QLP-solvers can be selected to create hybrid QLP-solvers. The selected QLP-solvers use both the real and surrogate objective functions. In addition, all population-based solvers use a single-solution based solver on the tail end in order to settle on a local optima with respect to all defined neighborhoods. In all of the following solvers, the subscript H is added to the solver's name if it uses (in some capacity) the surrogate objective function $f_{obj,u}$. The following QLP-solvers are studied in this research effort:

1. VND_{H}-based QLP-solver
    - This QLP-solver employs Algorithm 18 with $\gamma = \text{True}$. The VND_{H}-based solver is classified as a single-solution based solver. This solver first runs a round of VND with the surrogate $f_{obj,u}$ to find a locally optimal solution s^*. Then, a second round of VND is run with the true objective function f_{obj} with s^* as the initial solution.
2. VND-based QLP-solver

- This QLP-solver employs Algorithm 18 with $\gamma = \text{False}$. The VND-based solver is also classified as a single-solution based solver. This solver runs a single round of VND with the true objective function f_{obj}. The initial solution is randomly selected from the search space.

3. $BRKGA_H$+VND_H-based QLP-solver
    - This QLP-solver first uses a BRKGA (configuration defined in Section 3.3.4.2), which uses the surrogate, to find a solution s^*. Here, s^* is the highest-fitness solution found by the BRKGA. Then, s^* is provided as the initial solution to the VND_H-based QLP-solver. The $BRKGA_H$+VND_H-based solver is classified as a hybrid solver.
4. ES+VND-based QLP-solver
    - This QLP-solver first uses ES (configuration defined in Section 3.3.4.3), which uses the true objective function, to find a solution s^*. Here, s^* is the highest-fitness solution found be ES. Then, s^* is provided as the initial solution to the VND-based QLP-solver. The ES+VND-based solver is also classified as a hybrid solver.

3.4 Performance Analysis

Section 3.4 presents the process used to evaluate the effectiveness and efficiency of the devised meta-based QLP-solvers. In this performance analysis, the meta-based QLP-solvers are compared against SOTA QLP-solvers. Research question #1 asks "how effective and efficient are various metaheuristic algorithms at finding high-quality solutions to the QLP?" A solution is of high-quality if it yields a transpiled circuit which, when executed, yields the correct state-vector as frequently as possible, in the presence of gate and coherence errors. Since the QLP is only a subproblem of QPT, and solution quality is measured based on the output of the resultant

transpiled circuit yielded by a QLP-solver, the meta-based QLP-solvers must be integrated into transpilers.

Section 3.4.1 explains how the meta-based QLP-solvers are integrated into Qiskit's transpilers. Section 3.4.2 presents the benchmark QLP-solvers used in this research effort to evaluate their effectiveness and efficiency. Section 3.4.3 defines this research effort's experimental design. Finally, Section 3.4.4 defines metrics to analyze the effectiveness and efficiency of the meta-based QLP-solvers.

3.4.1 Integrating QLP-Solvers into Qiskit's Transpiler

The Qiskit transpiler, defined in Section 2.4.3, is composed of a series of transpiler passes (encapsulated in a $PassManager$) that transform an input circuit and optimize it to run on a specified backend QC. In the QST_3, the QLP-solver is broken into two transpiler passes: the $DenseLayout$ pass and the $StochasticSwap$ pass. A QLP-solver, by itself, does not create a circuit that is executable on quantum hardware. Thus, in order to analyze the effectiveness of the proposed QLP-solvers compared to Qiskit's (and address the posed research questions), each QLP-solver must be integrated into a $PassManager$ which performs all appropriate transpilation steps to create an executable circuit for a given backend QC.

In this experimental design, all meta-based transpilers are derived from the QST_0. The QST_0, by default, only includes gate decomposition passes, a $layout_{method}$ pass, and a $routing_{method}$ pass. The passes corresponding to a QLP-solver ($layout_{method}$ and $routing_{method}$ passes) are replaced by a meta-based QLP-solver. As such, the only experimental variable is the selected QLP-solver. As discussed in Section 3.3.6, this research effort considers the following meta-based QLP-solvers:

1. VND_H-based QLP-solver.

- Variable Neighborhood Descent-based QLP-solver; uses both the surrogate and true objective functions.

2. $BRKGA_H$+VND_H-based QLP-solver.
    - Biased Random-Key+Variable Neighborhood Descent-based QLP-solver; the BRKGA uses the surrogate only and the VND_H uses both the surrogate and true objective functions.
3. VND-based QLP-solver.
    - Variable Neighborhood Descent-based QLP-solver; uses only the true objective function.
4. ES+VND-based QLP-solver.
    - Evolution Strategies+Variable Neighborhood Descent-based QLP-solver; both ES and the VND use only the true objective function.

Each meta-based QLP-solver is integrated into the QST_0. Let $QST_0(QLP_{solver})$ define the QST_0 with the $layout_{method}$ and $routing_{method}$ passes replaced by QLP_{solver}. By replacing the $layout_{method}$ and $routing_{method}$ passes in the QST_0 with the meta-based QLP-solvers, the following meta-based transpilers are created:

1. $QST_0(VND_H)$
2. $QST_0(BRKGA_H + VND_H)$
3. $QST_0(VND)$
4. $QST_0(ES + VND)$

3.4.2 Benchmark State of the Art QLP-Solvers

To make a fair comparison of the meta-based QLP-solvers versus Qiskit's SOTA QLP-solvers, all of Qiskit's SOTA QLP-solvers are also integrated into the QST_0 (as done for the meta-based transpilers). Qiskit's SOTA transpiler, the QST_3, has a QLP-solver composed of the

$DenseLayout$ and $StochasticSwap$ passes. However, these two passes are not necessarily (or even claimed to be by IBMQ) superior to other $layout_{method}$ and $routing_{method}$ passes available in Qiskit. As such, the QST_0 is augmented with other $layout_{method}$ and $routing_{method}$ passes in order to test against all SOTA options for QLP-solvers available in Qiskit.

Let $QST_0(layout_{method}, routing_{method})$ define the QST_0 with $layout_{method}$ pass and a $routing_{method}$ pass. For this research effort, the following benchmark transpilers are created via augmentations of the QST_0 and are used for performance analysis of the meta-based transpilers:

1. $QST_0(Dense, Stochastic)$
2. $QST_0(NoiseAdaptive, Lookahead)$
3. $QST_0(Sabre, Sabre)$

The first includes the base QLP-solver of the QST_3. The second includes a $NoiseAdaptive$ initial mapping procedure and $Lookahead$ qubit routing method. The $NoiseAdaptive$ (NA) pass is derived from SOTA work by Murali et al. [12], and the $Lookahead$ (LA) pass is based on SOTA work by Zulehner et al. [11]. This configuration was also a benchmark in Niu et al. [41]. Finally, the last includes the $Sabre$ initial mapping and qubit routing methods of Li et al. [18]. Note that the $TrivialLayout$ and $Basic$ routing method from Section 2.4.3 are not included, as they do not employ any optimization techniques, and thus generally perform worse than available alternatives.

3.4.3 Experimental Design

Defining the experimental design begins with defining the goals of the research effort. In Section 3.3.3, quality of solutions and computational effort requirements are presented for the meta-based QLP-solvers. Since each meta-based QLP-solver is integrated into a meta-based transpiler, the requirements of Section 3.3.3 are modified as follows:

1. Solutions obtained from the meta-based transpilers should yield transpiled circuits that return the correct state-vector more often than the circuits generated by benchmark transpilers.
2. The meta-based QLP-solvers' execution time is significantly less than the period with which IBM updates calibration data.

Next, instances are selected to compare the effectiveness and efficiency of the meta-based transpilers to those of the benchmark transpilers. These instances are of varying difficulty. For backends, difficulty is imposed by testing against QCs with varying numbers of qubits. Another factor in backend difficulty is connectivity. All available IBM backends IBM have limited connectivity between their physical qubits. For circuits, difficulty is imposed by varying the depth (i.e. number of layers) and entanglement requirements (i.e. number of C_{NOT} operations).

Test Instance Backends: Backends are selected based on the number of physical qubits and connectivity between physical qubits. Backend (1) has 5 physical qubits and a nearest-neighbor topology. Backend (2) has 15 physical qubits and a grid-like topology. Both backends have limited connectivity between their physical qubits and are noisy (i.e. varying fidelities of 1-qubit and 2-qubit gates, limited T_1 and T_2 times). Figure 30 shows the topologies of the selected backends.

1. IBMQ Yorktown (5-Qubit QC)
2. IBMQ Melbourne (15-Qubit QC)

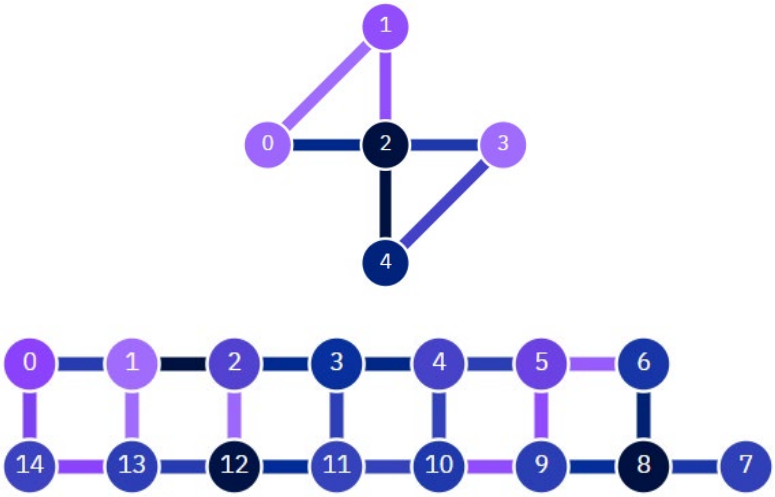


Figure 30: IBMQ Yorktown (top) and IBMQ Melbourne (bottom) topologies. Reproduced from "IBM Quantum Experience" [16].

Test Instance Circuits: Circuits are chosen based on varying depth. Also, Zulehner and Willie claim SU(4) circuits are a "worst-case" scenario for QPT due to high entanglement requirements [42], so all selected test circuits have such requirements. Moreover, test circuits should be easy to analyze, in the sense that when executed on an ideal QC, they have one correct state-vector output consisting of a single computational basis state (i.e. not a superposition state). In this research, the correct state-vector is referred to as a *truth-vector*. The following circuit classes are selected:

1. Quantum Fourier Transform (qft-x): The truth-vector of each instance is $(\bigotimes_{i=1}^{x-1} |0\rangle)|1\rangle$. This circuit class is chosen as it exhibits high entanglement requirements. In addition, the qft is a fundamental component of many other quantum algorithms, such as Shor's factoring algorithm [43].
2. Bernstein-Vazirani (bv-x): The truth-vector of each instance is $\bigotimes_{i=1}^{x} |1\rangle$. This circuit class is chosen as it illustrates challenging entanglement requirements; namely, hub-and-spoke entanglement [19]. In hub-and-spoke entanglement, one logical qubit is entangled with every

other logical qubit. With the chosen truth-vector, each measured qubit should be in the state $|1\rangle$. Holding such a truth-vector makes this circuit even more difficult to execute, as T_1 errors decay each $|1\rangle \rightarrow |0\rangle$.

Breakdown of Individual Test Cases: Since the QLP has inputs of both a backend and a circuit, individual test cases are created by selecting $(test\ circuit, backend)$ pairs from the Cartesian product $test\ circuits \times test\ backends$. Table 17 enumerates the test cases performed in this study. As stated previously, let L be the number of layers in the circuit, M be the number of logical qubits used in the circuit, and N be the number of physical qubits on the backend QC.

Backend Name	Circuit Name	L	M	N	Number of Decision Variables
IBMQ Yorktown	bv-2	4	2	5	20
IBMQ Yorktown	bv-3	5	3	5	25
IBMQ Yorktown	bv-5	7	5	5	35
IBMQ Yorktown	qft-3	15	3	5	75
IBMQ Melbourne	bv-5	7	5	15	105
IBMQ Yorktown	qft-5	31	5	5	155
IBMQ Melbourne	bv-8	10	8	15	150
IBMQ Melbourne	qft-3	15	3	15	225
IBMQ Melbourne	bv-15	17	15	15	255
IBMQ Melbourne	qft-6	39	6	15	585

Table 17: Test cases to be evaluated in this study.

As stated by Talbi, "[t]he selection of input instances to evaluate a given metaheuristic [must] be chosen carefully. The set instances must be diverse in terms of size of instances, their difficulties, and their structure" [6]. The test cases in Table 17 are diverse in terms of size of instances (as indicated by the number of decision variables). The number of decision variables also correlates with the problem-instance difficulty. That is, optimization problems with more decision variables are typically more difficult to solve. Finally, most test circuits are of the most difficult structure to optimize (many C_{NOT} gates in circuits and backends with limited connectivity between their qubits). In addition, all test cases are executed on noisy backend simulators and yielded the correct state-vector an observable number of times.

Experiments: The ES-based transpiler is executed via HPCs. The elapsed time between job insertion into the queue and retrieval of results is on the order of days (rendering the calibration data obsolete; see Section 3.4.4 for more details). In addition, to execute quantum programs on IBM's QCs, the circuit must be submitted as a job. At the time of this writing, the queue times for the selected backends exceed the calibration window defined in Section 3.4.4. Accordingly, in lieu of testing the transpiled circuits directly on quantum hardware, Qiskit's $NoiseModel$ instances are used to evaluate transpiler performance. A $NoiseModel$ instance captures the main parameters of the objective functions, as well as the cost functions of all SOTA QLP-solvers being used as the basis for comparison. These parameters include single-qubit error rates, two-qubit error rates, readout error rates, T_1 error rates, T_2 error rates, connectivity of backend QC, and the basis gate-set of the backend QC.

First, a $NoiseModel$ instance is created for each test case backend QC. Then, for each test case in Table 17, the test case is transpiled via each meta-based transpiler and each benchmark transpiler (in both cases using the test case's associated $NoiseModel$ instance for its backend). Finally, the resultant transpiled circuits from the meta-based transpilers and benchmark transpilers are executed on a simulated backend (again with the test case's associated $NoiseModel$ instance for its backend), and results are compared using metrics defined in Section 3.4.4. The testing procedure is defined in Algorithm 20.

Algorithm Testing procedure for meta-based transpilers versus Qiskit.

Inputs: All test cases from Table 17 (set T).

$META\_transpilers = [QST_0(VND), QST_0(VND_H), QST_0(BRKGA_H + VND_H), QST_0(ES + VND)]$;
$BEN\_transpilers = [QST_0(Dense, Stochastic), QST_0(NoiseAdaptive, Lookahead), QST_0(Sabre, Sabre)]$;

/* Generate noise models for each backend a priori */
$noise\_models$ = {} ; /* Dictionary to hold noise models for each backend */
For $circuit, backend$ in T **Do**
  $noise\_models[backend] = NoiseModel(backend)$;
End For

/* Transpile and run each test case against appropriate noise model */
For $circuit, backend$ in T **Do**
  $noise\_model = noise\_models[backend]$; /* get noise model for backend */

  /* Transpile and execute circuit with all $T_\beta \in BEN\_transpilers$ */
  $results\_ben$ = [] ; /* List to results from $BEN\_transpilers$ */
  For T_β in $BEN\_transpilers$ **Do**
    $trans\_circ\_ben = T_\beta.transpile(circuit, noise\_model)$;
    $result\_ben = execute(trans\_circ\_ben, noise\_model)$;
    $results\_ben.append(result\_ben)$;
  End For

  For T_α in $META\_transpilers$ **Do**
    /* Transpile circuit with T_α */
    $trans\_circ\_meta = T_\alpha.transpile(circuit, noise\_model)$;

    /* Execute transpiled circuits on simulator with noise model */
    $results\_meta = execute(trans\_circ\_meta, noise\_model)$;

    /* Compare results via measurements defined in Section 3.4.4 */
  End For
End For

Algorithm 20: Testing procedure for meta-based transpilers versus Qiskit.

The test procedure in Algorithm 20 is carried out three times because all meta-based QLP-solvers are stochastic (as well as some benchmark transpilers). Three rounds provide more data points to statistically analyze the effectiveness of all transpiler options and is a computationally feasible number of rounds to carry out the test procedure in Algorithm 20. After three trials, only the highest-fidelity solutions attained by each transpiler are analyzed in Chapter IV because

highest-fidelity is favored over average-fidelity in the assessment of effectiveness of various QLP-solvers in this research effort.

Highest-fidelity is favored because in the assessment of effectiveness, this research seeks to determine if meta-based transpilers are capable of outperforming benchmark transpilers. Thus, the potential efficacy of meta-based transpilers is contingent on their ability to achieve higher-quality solutions than benchmark transpilers. Average-fidelity assessment is a step beyond this research effort, as it does not make sense to improve the average-case if the best-case fidelities of the meta-based transpilers do not surpass those of the benchmark transpilers.

3.4.4 Measurements

In this section, metrics are defined to quantify the effectiveness and efficiency of a transpiler. First, effectiveness metrics are defined in sub-section "Quality of Solution Measurements". Then, efficiency metrics are defined in sub-section "Computational Effort Measurements".

Quality of Solution Measurements: The quality of a solution depends on the state-vectors returned by the circuit being measured. By the design of the experiment, the ideal output of each of the circuits used in these experiments is a single computational basis state. Thus, for a circuit with x-qubits, exactly one of the 2^x computational basis states is the correct answer. First, for all test cases, 1024 shots are executed on the backend QC of the transpiled circuit (that is, the same circuit is executed 1024 times for a given test case). Let C be the set of all test cases in Table 17. Then, for a test case $c \in C$ and a transpiler $T \in$ {meta-based transpilers, benchmark transpilers}, let $count(T, c)$ be the number of shots that return the correct state-vector for the circuit of test case c. The success rate of the resultant transpiled circuit is defined as follows:

$$success_{rate}\big(count(T,c)\big) = \frac{count(T,c)}{1024}$$

The $success_{rate}$ represents the fidelity of the resultant transpiled circuit generated by T for test case c.

Computational Effort Measurements: The computational effort depends on how long the meta-based QLP-solvers take to find high-quality solutions to QLP-problem instances. Even though the transpilers do perform other transpiler passes, the QLP-solver pass is the most time-intensive subroutine of QPT, and all other transpiler passes (consisting only of gate decomposition passes for the studied transpilers) have negligible execution times. Since Qiskit uses heuristics-based methods to find solutions to the QLP, the meta-based QLP-solvers are anticipated to execute much more slowly than Qiskit's QLP-solvers.

While the meta-based QLP-solvers are anticipated to transpile much more slowly than Qiskit's QLP-solvers, the calibration data (provided by IBM) that the meta-based QLP-solvers rely on is updated approximately every 24 hours according to Wilson et al. [44]. As such, the meta-based QLP-solvers must find a solution to the QLP within this $calibration_{window}$; otherwise, the solution found is no longer relevant since the noise-data for the QC is likely different. Let $transpilation_{time}$ be the amount of time, in milliseconds (ms), a given meta-based QLP-solver takes to find a good solution to the QLP and $calibration_{window} = 24 \text{ hours} = 86{,}400{,}00 \text{ ms}$. Now, a relative computational effort metric is defined as follows:

$$comp_{effort} = \frac{transpilation_{time}}{calibration_{window}}$$

3.5 Fitness Landscape Analysis of the QLP

This section defines fitness landscapes induced by the neighborhood functions and representations of the QLP. Sections 3.5.1 and 3.5.2 define the phenotypic and the genotypic fitness landscape for the QLP, respectively. Then, Section 3.5.3 provides the fitness landscape

analysis metrics used in this research effort to analyze QLP fitness landscapes. In addition, Section 3.5.3 provides the fitness landscape analysis procedure.

3.5.1 Phenotypic Landscape

The phenotypic landscape $\mathcal{L}_p$ of the QLP is defined via sequences of L permutations of N discrete-valued elements (i.e. all $\lambda_i \in M_h$). Each such sequence is a configuration of $\mathcal{L}_p$. Two distinct phenotypic landscapes are induced by considering both the true and surrogate objective functions f_{obj} and $f_{obj,u}$, respectively. The landscape is analyzed in the context of the neighborhood function N_2 from Section 3.3.4.

N_2 was selected because it represents a strong locality neighborhood used frequently by the VND as well as (implicitly) by the BRKGA and ES (during evolutionary operations). In addition, the size of N_2 is relatively small (space complexity $O(L \cdot N^2)$). Moreover, in most permutation-based problems, 1-exchange neighborhoods are frequently used to explore parameter space. Next, let $d(s_i, s_j)$ denote the distance between solutions s_1 and s_2, where the distance is calculated as the number of displaced elements between s_1 and s_2. Figure 31 illustrates this distance metric.

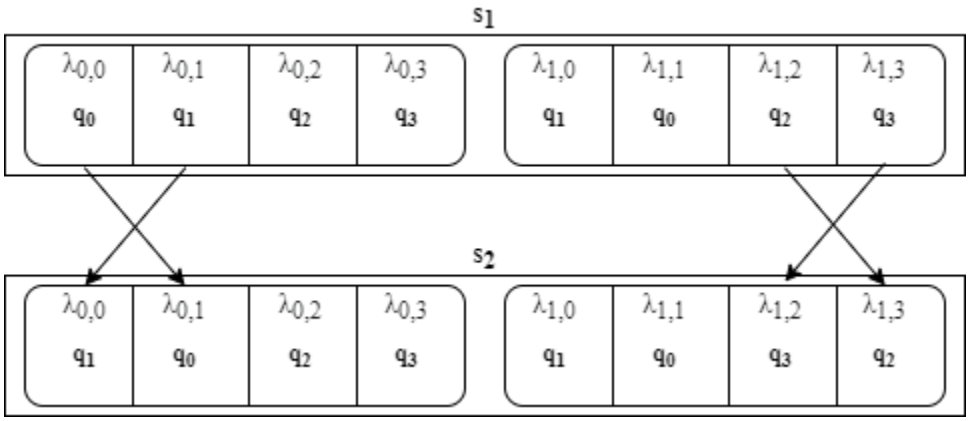


Figure 31: Distance metric in $\mathcal{L}_p$. In this figure, $d(s_1, s_2) = 4$, as four elements are displaced between s_1 and s_2.

3.5.2 Genotypic Fitness Landscape

The genotypic fitness landscape $\mathcal{L}_g$ of the QLP is defined by L lists of N real-valued variables where the value of each variable is in range [0,1]. That is, the set X of configurations of $\mathcal{L}_g$ is all valid assignments of each decision variable in the continuous space $\mathbb{R}^{L \cdot N}$ (when all $\lambda_i \in M$ are flattened into a 1-dimensional list). Similar to the $\mathcal{L}_p$, this fitness landscape also uses the objective functions f_{obj} and $f_{obj,u}$. Given a solution s, the neighborhood $N(s)$ is the ball with center s and radius ϵ for some $\epsilon > 0$. The distance metric between solutions $s, s' \in \mathbb{R}^{L \cdot N}$ is defined by the Euclidean norm

$$d(s, s') = \sqrt{\sum_{i=0}^{(L \cdot N)-1} \left(s_i' - s_i\right)^2}$$

3.5.3 Fitness Landscape Analysis Procedure

The analysis of $\mathcal{L}_p$ is carried out via enumeration of the entire search space (for smaller problem-instances) and sampling of the search space (for larger problem-instances). In this research effort, only $\mathcal{L}_p$ is analyzed. While analysis of $\mathcal{L}_g$ would provide insight, it is beyond the scope of this research effort.

To learn about the topology of $\mathcal{L}_p$, MDS (described in Section 2.6.2) is used to plot the fitness landscape on a 2-dimensional plot. Since MDS is operationally intensive, MDS is only used to analyze small problem-instances. Specifically, MDS is used to analyze the bs-1 circuit (shown in Figure 3 of Section 2.2.3) transpiled to the IBMQ Yorktown backend. In addition, objective functions f_{obj} and $f_{obj,u}$ are plotted. This circuit was chosen as the total number of permutations for this circuit transpiled to the IBMQ Yorktown is $(N!)^L = (5!)^2 = 14{,}400$ (a computationally feasible problem-size for MDS).

While plotting the fitness landscape is interesting, other metrics exist to analyze fitness landscapes that do not require complete enumeration of the search space. Malan and Engelbrecht provide many metrics that can be used to analyze a fitness landscape [26]. In this study, the following metrics are selected to analyze $\mathcal{L}_p$:

Metric	Description	Technique(s) to Analyze
Ruggedness	This metric is used to analyze the number and distribution of local optima in the search space. Rugged landscapes have many local optima, while smooth landscapes have few local optima.	Autocorrelation function of Weinberger [28]
Evolvability	This metric broadly measures the capability of a search process to move to a place in the fitness landscape of better fitness.	Fitness Cloud Technique of Verel et al. [29] Fitness-Probability Cloud and Accumulated Escape Probability Techniques of Lu et al. [30]
Epistasis	This metric is used to analyze the amount of interaction among the decision variables. The more rugged the fitness landscape, the higher the epistasis [6]. Thus, ruggedness metrics can also be used to analyze epistasis.	Autocorrelation function of Weinberger
Fitness Distribution	This metric is used to analyze how fitness values are distributed across the search space.	Density of States Technique of Rose et al. [31]

Table 18: Metrics used in this research effort to analyze fitness landscapes.

Details on the techniques selected and algorithms employed for each technique are in Section 2.6.1. Next, Table 19 provides the test cases selected for analysis of $\mathcal{L}_p$. The test cases are selected based on the diversity of problem-instances (e.g. number of decision variables and circuit classes) and computational effort needed to carry out the fitness landscape analysis techniques.

Backend Name	Circuit Name	L	N	Number of Decision Variables
IBMQ Yorktown	bv-2	4	5	20
IBMQ Yorktown	bv-5	7	5	35
IBMQ Yorktown	qft-3	15	5	75
IBMQ Yorktown	qft-5	31	5	155

Table 19: Selected test cases to analyze fitness landscape $\mathcal{L}_p$ in this research effort.

Algorithm Testing procedure for fitness landscape analysis.

Inputs: All test cases from Table 19 (set T).

$n = 500$; /* number of sample points */

/* Use MDS to plot fitness landscapes of small problem */
$perms$ = get all permutations of bs-1 solution transpiled to IBMQ Yorktown ;
D = compute distance matrix for all permutations ;

$points = MDS(D, dimensions = 2)$;
C_1, C_2 = create distinct canvases to plot the true and surrogate fitness landscapes ;
For $idx, point$ in $enumerate(points)$ **Do** /* $point = (x_i, y_j)$ */
  $plot$ point $\left(x_i, y_j, f_{obj}(perms[idx])\right)$ on C_1;
  $plot$ point $\left(x_i, y_j, f_{obj,u}(perms[idx])\right)$ on C_2;
End For

/* Perform fitness landscape analysis techniques on test cases from Table 19 */
$obj\_functs = [f_{obj}, f_{obj,u}]$;
For $backend, circuit$ in T **Do**
  For f in obj_functs **Do**
    /* 1. Perform autocorrelation function test */
    $p = autocorrelation\_f(backend, circuit, f, N_2, n)$;

    /* 2. Perform fitness distribution test */
    $fitness\_distribution(backend, circuit, f, N_2, n)$;

    /* 3. Perform fitness cloud test */
    $fitness\_cloud(backend, circuit, f, N_2, n)$;

    /* 4. Perform fitness-probability cloud and accumulated escape probability tests */
    $e_p = fitness\_prob\_cloud(backend, circuit, f, N_2, n)$; /* e_p is the escape probability */

    /* Analyze results */
  End For
End For

Algorithm 21: Testing procedure for fitness landscape analysis.

Algorithm 21 provides the testing procedure for fitness landscape analysis. In this procedure, first MDS is used to plot the fitness landscapes of the bs-1 transpiled to the IBMQ Yorktown. Next, the selected landscape analysis techniques are performed on each selected test case in Table 19. Since a fitness landscape is induced by a representation (unchanged, always phenotypic), a neighborhood function (unchanged, always N_2), and an objective function (varies, using f_{obj} and $f_{obj,u}$), the techniques must be carried out for both objective functions (as each induces a distinct fitness landscape). Each iteration of the innermost for loop yield plots for each technique as well as an autocorrelation value (p) and an escape probability (e_p).

3.6 Summary

This chapter has presented the methodology used to analyze the research questions presented in Chapter I. First, in Section 3.3 the QLP is mathematically modeled and integrated into various metaheuristic algorithm domains. Section 3.4 then defines the performance analysis procedures and metrics this research effort uses to evaluate the effectiveness and efficiency of the meta-based QLP-solvers. Finally, Section 3.5 outlines the fitness landscape analysis procedure for this research effort used to analyze the induced fitness landscapes of various QLP problem-instances.

IV. Results and Analysis

4.1 Overview

The research results are presented and analyzed in this chapter. This chapter is divided into four sections. In Section 4.2, the quality of solutions is presented and analyzed for the devised meta-based QLP-solvers versus SOTA QLP-solvers. In Section 4.3, the computational effort of each meta-based QLP-solver is presented and analyzed. In Section 4.4, the results of all fitness landscape analysis techniques are presented and analyzed. Table 20 provides the abbreviated names given to each transpiler for the remaining chapters. The first three transpilers in Table 20 are Qiskit's and the remaining are meta-based transpilers devised in this research effort (see Sections 3.4.1 and 3.4.2).

Transpiler	Abbreviated Name	Description
$QST_0(DS)$	DS	Benchmark transpiler
$QST_0(NA)$	NA	Benchmark transpiler
$QST_0(Sabre)$	Sabre	Benchmark transpiler
$QST_0(VND_H)$	VND-H	Meta-based transpiler
$QST_0(BRKGA_H + VND_H)$	BRKGA-H+VND-H	Meta-based transpiler
$QST_0(VND)$	VND	Meta-based transpiler
$QST_0(ES + VND)$	ES+VND	Meta-based transpiler
$QST_0(BRKGA_H)$	BRKGA-H	Meta-based transpiler
$QST_0(ES)$	ES	Meta-based transpiler

Table 20: Abbreviated transpiler names.

Moreover, the names of the meta-based transpilers and the algorithms that they employ are used interchangeably in the remaining chapters. The only distinguishing factor between two given transpilers is the QLP-solver they employ. Also, note that any time success rate or fidelity are mentioned, these refer to the $success_{rate}$ outcome of a test case and transpiler.

4.2 Quality of Solutions

In this section, the quality of solutions is presented and analyzed for all test cases in Table 17. To collect results, the testing procedure from Section 3.4.3 is used, along with measurements defined in Section 3.4.4. Below, the $success_{rate}$ metric for each test case is presented in Figure

32. This metric provides an absolute comparison of the quality of solutions obtained by the meta-based and benchmark transpilers. The x-axis captures the test case and the y-axis captures the percentage of shots that returned the correct state-vector.

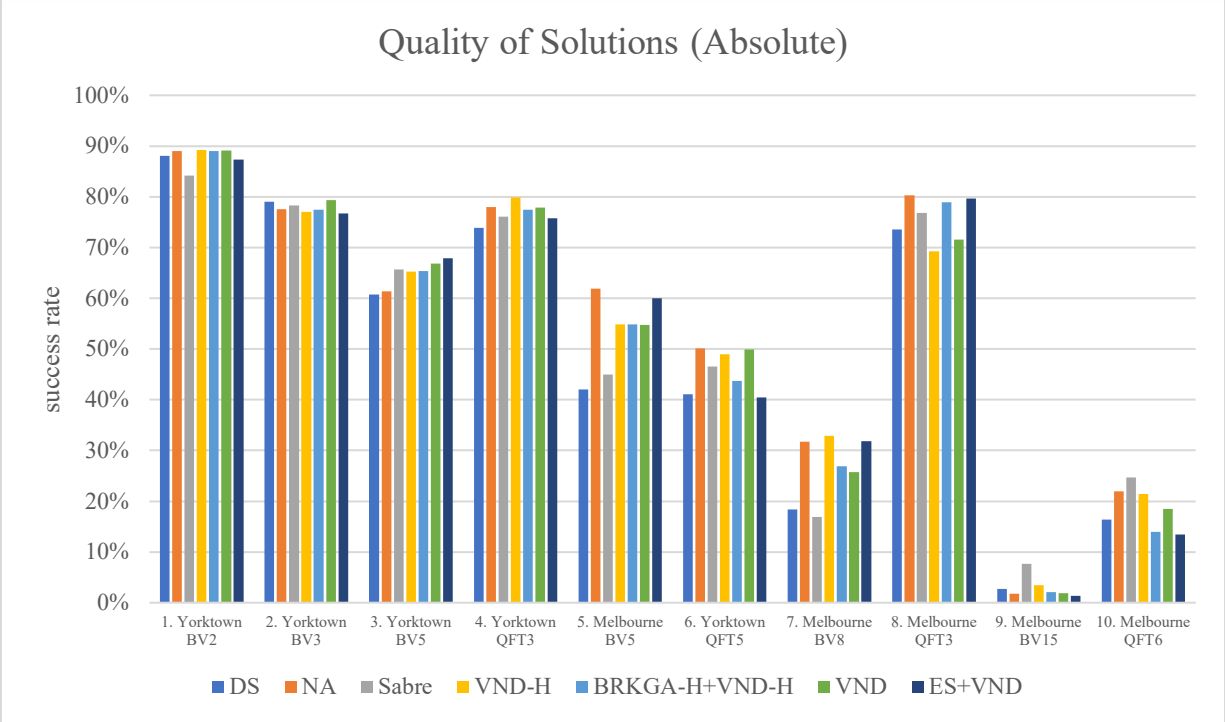


Figure 32: Quality of solutions (absolute) for all test cases in Table 17.

First, the results in Figure 32 suggest that in the first four test cases, meta-based transpilers and benchmark transpilers obtain similar-quality results. Consider Table 21, which presents the standard deviations of each test case.

Test Case #	Test Case	Standard Deviation
1	Yorktown BV2	0.0182
2	Yorktown BV3	0.0099
3	Yorktown BV5	0.0268
4	Yorktown QFT3	0.0193
5	Melbourne BV5	0.0732
6	Yorktown QFT5	0.0409
7	Melbourne BV8	0.0653
8	Melbourne QFT3	0.0432
9	Melbourne BV15	0.0216
10	Melbourne QFT6	0.0426

Table 21: Standard deviations of test cases.

Visual analysis of the results in Table 21 also indicate that the first four test cases have lower standard deviations than the latter six. To further investigate, the first four test cases and last six test cases are analyzed collectively. Test cases 1-4 have a mean standard deviation of approximately 2%, while test cases 5-10 have a mean standard deviation of approximately 5%. This means that for the latter test cases, the quality of solutions obtained by each transpiler vary more than the former test cases. This is likely because the latter test cases have much larger solution spaces, which causes the benchmark transpilers to make numerous globally sub-optimal steps and the meta-based transpilers to converge on low-quality local optima. The observed standard deviations indicate that test cases 1-4 are easier for the QLP-solvers to optimize than test cases 5-10.

Meta-based transpilers obtain equal or higher-quality solutions over all benchmark transpilers in 7 of the 10 test cases, which include challenging test cases such as the Melbourne BV8 and Melbourne QFT3. Table 22 breaks down the test cases in which each meta-based transpiler attained equal or higher-quality solutions than all benchmark transpilers.

Meta-Based Transpiler	Test Cases Attained Equal or Higher-Quality Solution Over All Benchmark Transpilers
VND-H	1, 4, 7
BRKGA-H+VND-H	1
VND	1, 2, 3, 4, 6
ES+VND	3, 7, 8

Table 22: Comparison of solution-quality attained by meta-based transpilers versus benchmark transpilers.

Results in Table 22 show that the VND performed best across all test cases out of the meta-based transpilers, followed by the VND-H and ES+VND. The BRKGA-H+VND-H performed poorly on most test cases, only attaining equal or higher-quality solutions than benchmark transpilers in the easiest test case. These results also show that the meta-based transpilers outperform all benchmark transpilers more frequently for the first four test cases. This also supports the claim that the latter test cases are much more challenging (at least for the meta-based transpilers).

While $success_{rate}$ is a useful metric to analyze each test case, it fails to show the relative effectiveness of each meta-based transpiler versus each benchmark transpiler. To analyze relative quality of solutions, let $T_\alpha \in$ {benchmark transpilers} and $T_\beta \in$ {meta-based transpilers}. For a given test case c, the relative performance of a meta-based transpiler T_α and a benchmark transpiler T_β is defined as follows:

$$success_{rel}\left(count(T_\alpha, c), count\left(T_\beta, c\right)\right) = \frac{success_{rate}\left(count\left(T_\beta, c\right)\right)}{success_{rate}\left(count(T_\alpha, c)\right)}$$

Next, the average relative success across all test cases of a given meta-based transpiler T_α in comparison to a given benchmark transpiler T_β is expressed as follows:

$$success_{rel}^{\Sigma}\left(T_\alpha, T_\beta\right) = \frac{\sum_{c \in C} success_{rel}\left(count(T_\alpha, c), count\left(T_\beta, c\right)\right)}{|C|}$$

Now, consider Table 23 which presents the relative effectiveness of each meta-based transpiler versus each benchmark transpiler.

Meta-Based vs. Benchmarks	VND-H	BRKGA-H+VND-H	VND	ES+VND
DS	+19%	+6%	+8%	+6%
NA	+7%	-6%	-5%	-8%
Sabre	+5%	-4%	-2%	-2%

Table 23: Quality of solutions (relative) for all test cases in Table 17. Read across then down (e.g. the VND-H obtains results that are on average 19% better than the DS).

The relative success metric shows that all meta-based transpilers obtained, on average, higher quality solutions than the DS. The BRKGA-H+VND-H, VND, and ES+VND all obtain lower-quality solutions than the NA and Sabre, on average. Interestingly, the VND-H outperforms all benchmark transpilers, with average solution-quality gains of 19% over DS, 7% over NA, and 5% over Sabre. This suggests that the VND-H obtains high-quality solutions more consistently than the benchmark transpilers.

To further analyze the quality of solutions, the number of layers inserted by all transpilers is used to gain insight into the local optima reached by each. Consider Figure 33.

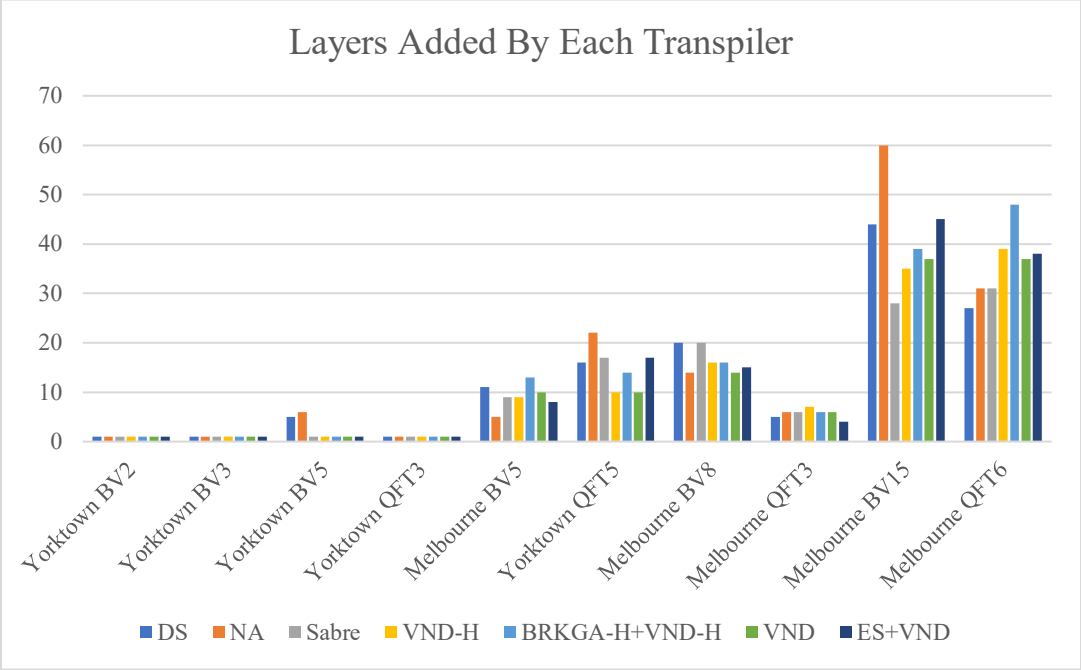
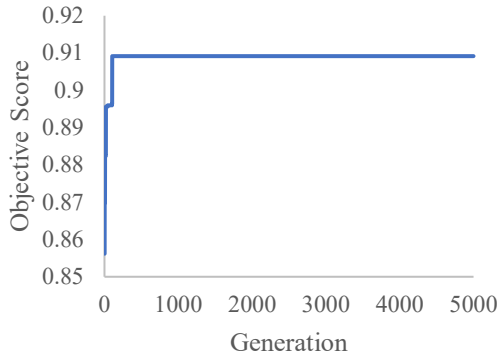
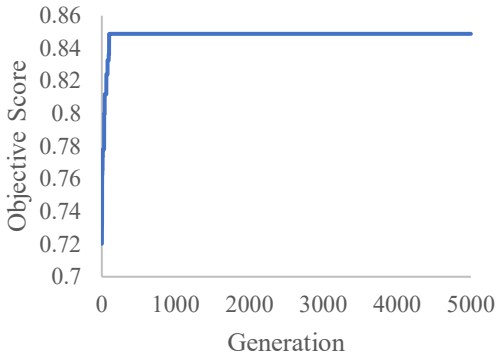
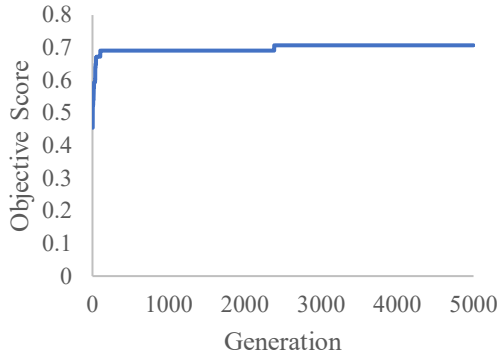
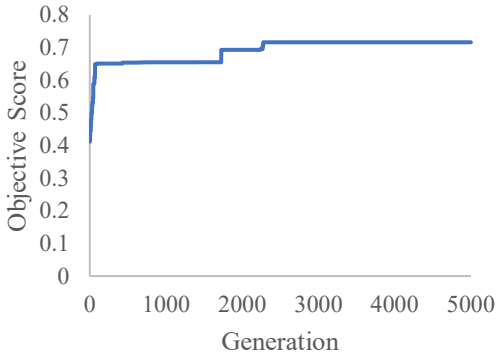
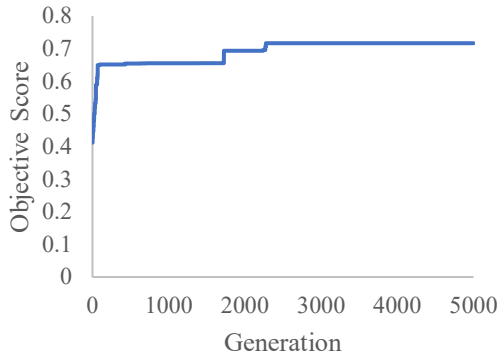
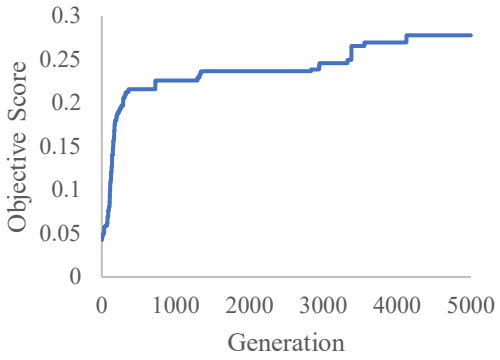


Figure 33: Number of layers added by each transpiler.

For the first four test cases, all transpilers add a small number of layers to the input circuit. Correspondingly, the yielded fidelities in Figure 32 are similar among all transpilers for the first four test cases. However, for the more challenging test cases, there is much higher variation in the number of layers added by each transpiler. The first four test cases have a mean standard deviation of 0.55, whereas the latter six test cases have a mean standard deviation of 4.55. This implies that in the first four test cases, the number of layers inserted by all transpilers differs, on average, by 0.55 layers. For the latter six test cases, the number of layers inserted by all transpilers differs, on average, by 4.55 layers.

When comparing the success rates of Table 22 and layers added, the Pearson correlation coefficient between the two data sets yields a value of -0.91. This statistical analysis verifies the hypothesis: higher $success_{rate}$ is correlated with a smaller number of layers added. As an example, the ES+VND adds 45 layers and has a $success_{rate}$ of 1% for the Melbourne BV15. On the same test case, the Sabre adds 28 layers and has a $success_{rate}$ of 8%. For the longer running circuits, adding more layers highly degrades $success_{rate}$ due to T_1 and T_2 errors. The effects of this are not as obvious for the easier test cases, as they are able to perform calculations within the coherence window of the devices' qubits.

The quality of solutions attained by the BRKGA-H+VND-H and ES+VND are further analyzed via trajectory information collected for both population-based algorithms. In Table 24 and Table 25, the trajectory information for the algorithms employed by the BRKGA-H and ES are presented, respectively. For the ES, the average standard deviations of the strategy parameters are also provided. Since both population-based solvers are executed multiple times per test case, only trajectory information for the runs which yielded the best results are shown.

Yorktown BV2	Yorktown BV3
	
Yorktown BV5	**Yorktown QFT3**
	
Melbourne BV5	**Yorktown QFT5**
	
Melbourne BV8	**Melbourne QFT3**

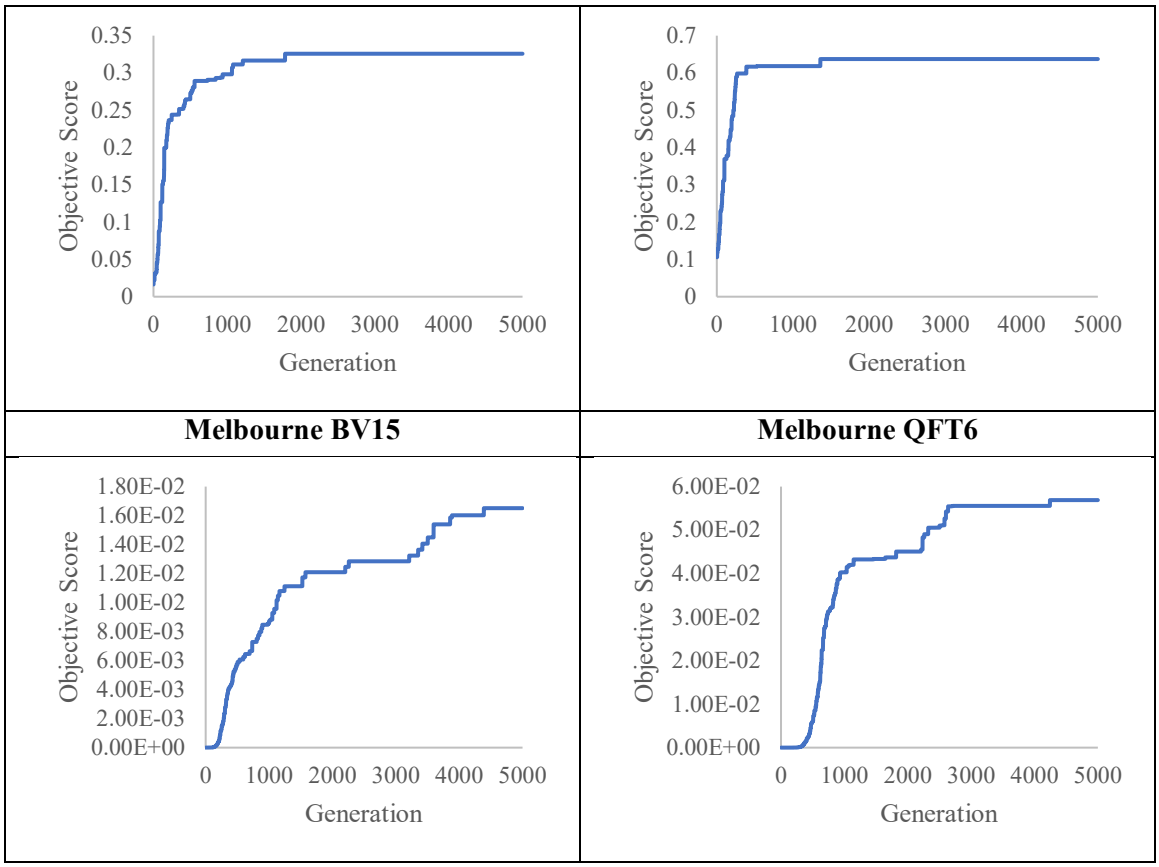
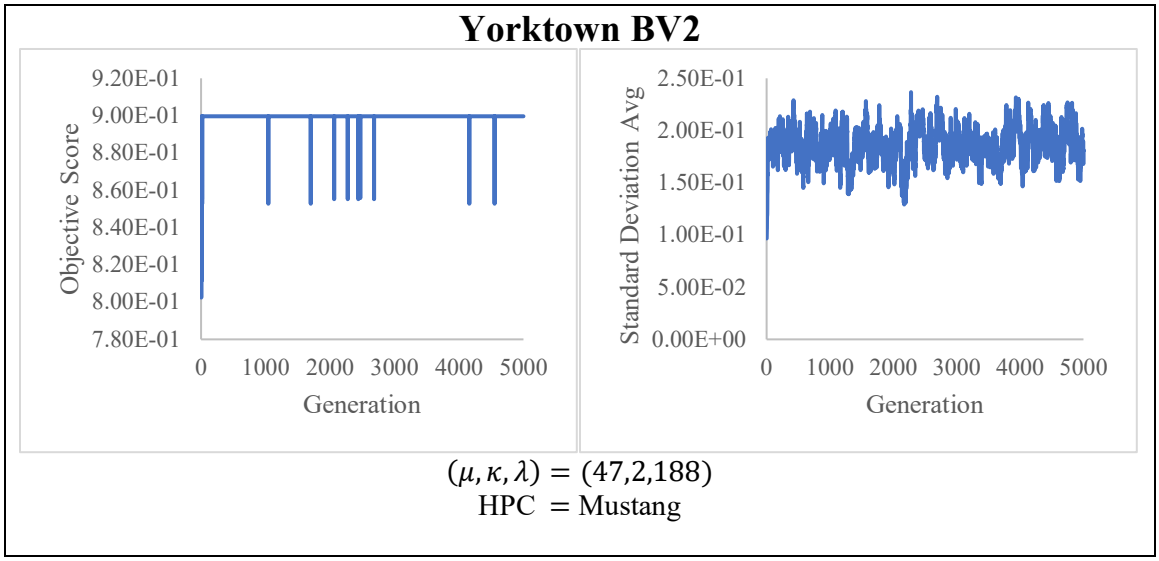
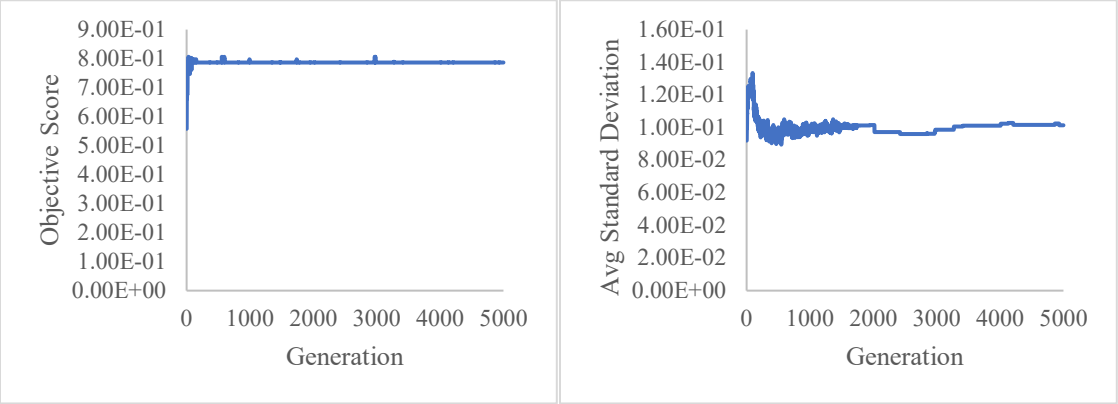


Table 24: BRKGA objective score trajectory data.

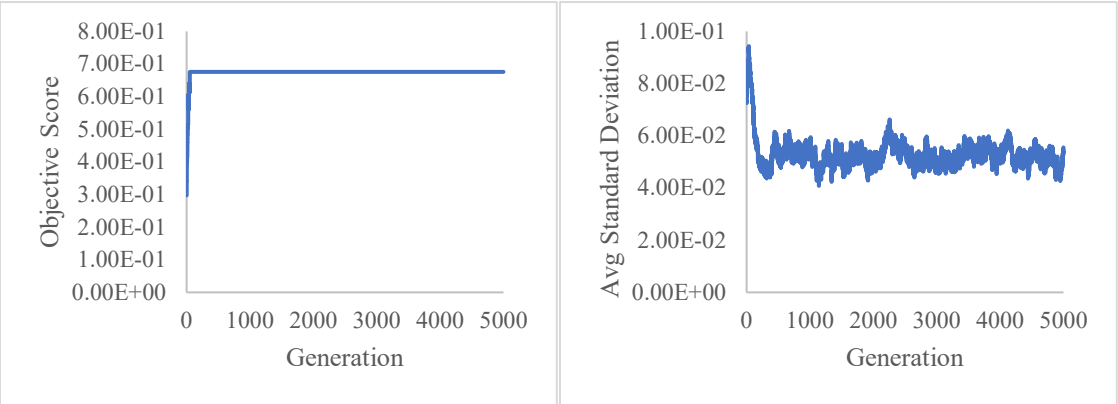


Yorktown BV3



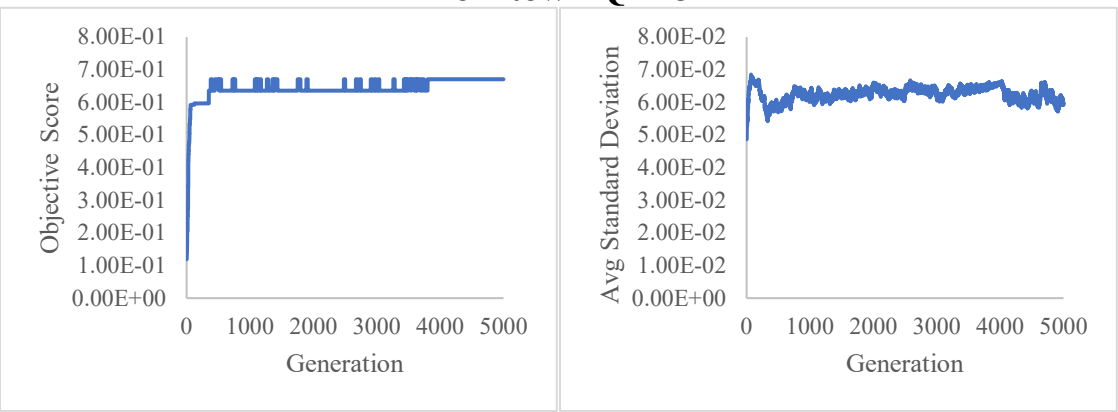
$(\mu, \kappa, \lambda) = (58,5,232)$
HPC = Gaffney

Yorktown BV5



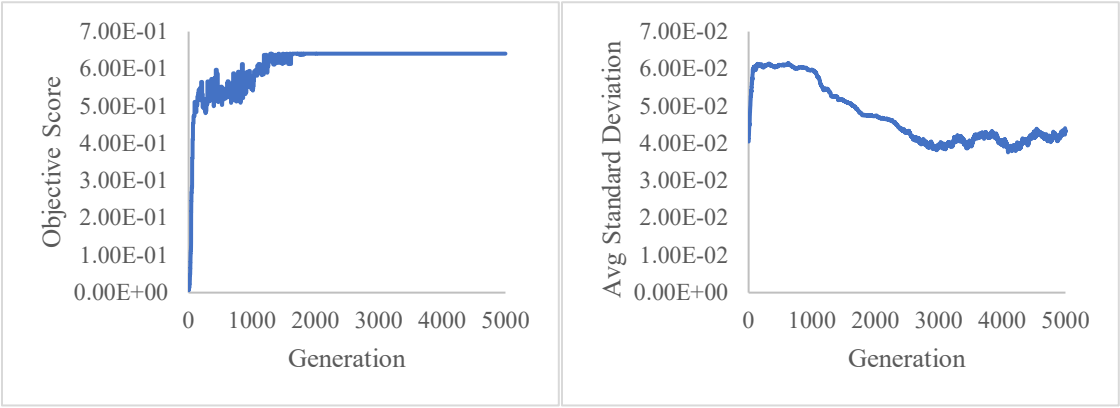
$(\mu, \kappa, \lambda) = (82,8,328)$
HPC = Gaffney

Yorktown QFT3



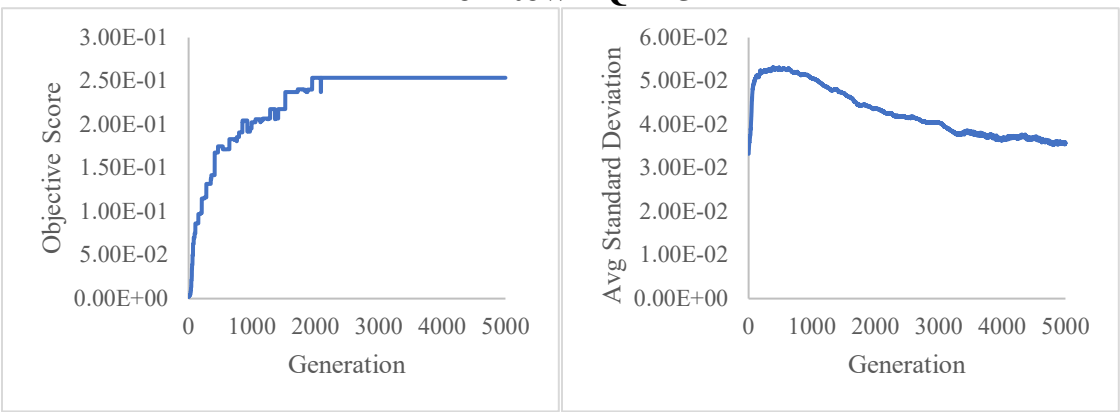
$(\mu, \kappa, \lambda) = (175,22,700)$
HPC = Mustang

Melbourne BV5



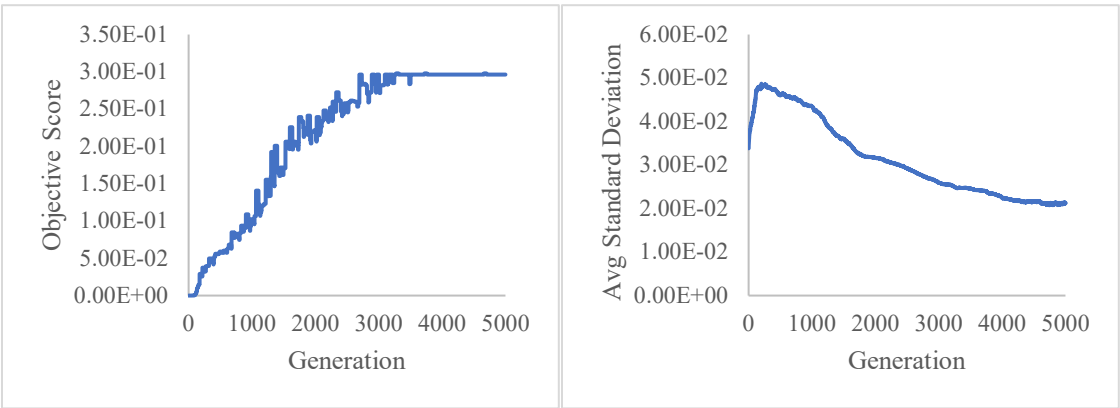
$(\mu, \kappa, \lambda) = (250,17,1000)$
HPC = Mustang

Yorktown QFT5



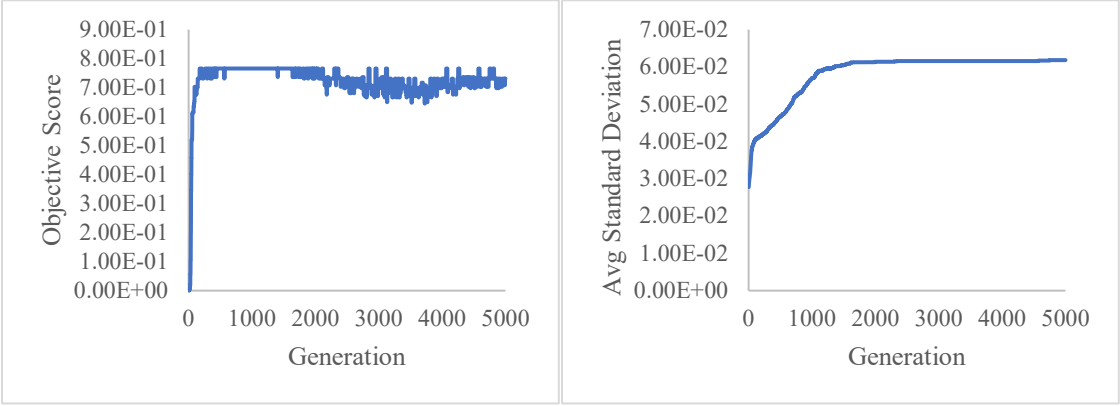
$(\mu, \kappa, \lambda) = (370,77,1480)$
HPC = Mustang

Melbourne BV8



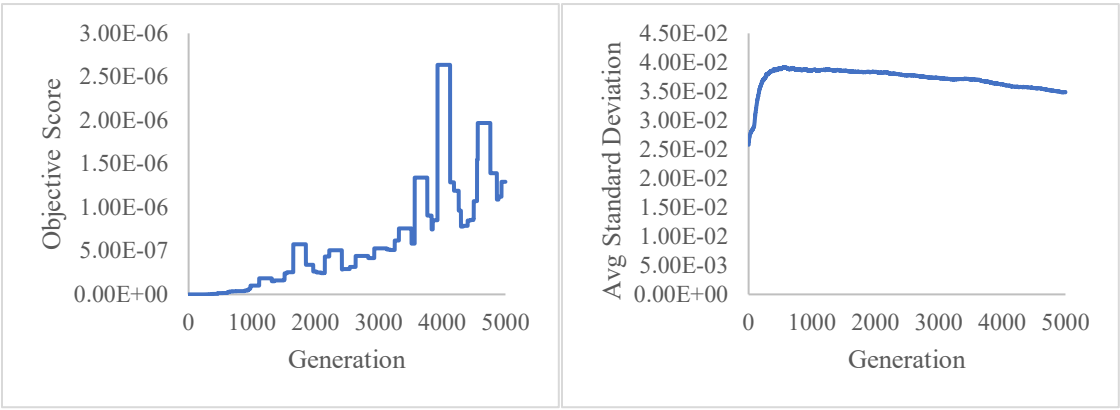
$(\mu, \kappa, \lambda) = (350,40,1400)$
HPC = Mustang

Melbourne QFT3



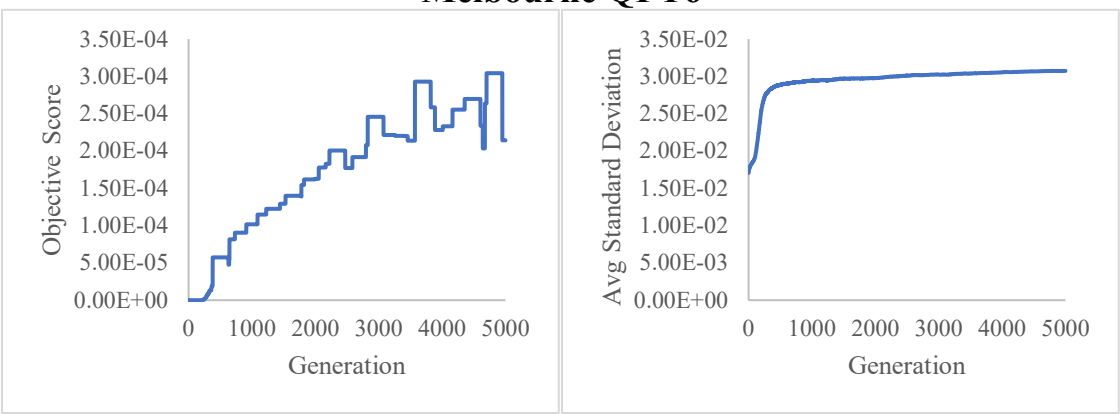
$(\mu, \kappa, \lambda) = (525,20,2100)$
HPC = Gaffney

Melbourne BV15



$(\mu, \kappa, \lambda) = (600,200,2400)$
HPC = Mustang

Melbourne QFT6



$(\mu, \kappa, \lambda) = (1400,250,5600)$
HPC = Gaffney

Table 25: ES objective score and standard deviation trajectory data.

The BRKGA-H and ES both use 5,000 generations for each test case. The BRKGA-H uses a population size of $L \cdot N$ while the ES uses a population size of $10 \cdot L \cdot N$. Further, the BRKGA-H uses the surrogate objective function $f_{obj,u}$ while the ES uses the true objective function f_{obj}. Accordingly, the objective scores from the BRKGA-H are anticipated to be higher than those from ES because the surrogate objective function is an upper-bound on the true objective function.

Convergence of the BRKGA-H is apparent when the fitness of the best-found solution does not increase for 2,500 generations (as defined in Section 3.3.5). Based on the objective trajectories, the BRKGA-H shows signs of convergence on all test cases except the Yorktown QFT5, Melbourne BV15, and Melbourne QFT6. In all other test cases, the BRKGA-H attains its highest fidelity solution within 2,500 generations. Consider Table 26, which compares the fidelities of the best-solutions found by the BRKGA-H and those of any other transpiler.

Test Case #	Test Case	BRKGA	Highest-Fidelity Solution Found
1	Yorktown BV2	88%	89%
2	Yorktown BV3	76%	79%
3	Yorktown BV5	67%	68%
4	Yorktown QFT3	66%	80%
5	Melbourne BV5	30%	62%
6	Yorktown QFT5	18%	50%
7	Melbourne BV8	5%	33%
8	Melbourne QFT3	33%	80%
9	Melbourne BV15	0%	8%
10	Melbourne QFT6	3%	25%

Table 26: BRKGA-H solution-fidelities versus best found.

The BRKGA-H shows signs of convergence on test cases 1, 2, 3, 4, 5, 7, and 8. For these test cases, the BRKGA-H attains solution-qualities that are on average 40% worse than the best found. This confirms that the BRKGA-H settles on poor local optima even when the algorithm appears to converge. For the test cases where convergence is not apparent, more generations of

the BRKGA-H are required to see where evolution stalls. Alternatively, further parameter tuning of the BRKGA-H's hyperparameters may yield better convergence and/or convergence on higher-fit optima.

Convergence in ES is apparent when the average standard deviation of all strategy parameters decreases towards zero. For most test cases, the average standard deviations indicate that the ES solver was unable to settle on a local optimum. Like the BRKGA-H, consider Table 27, which compares the fidelities of the best-solutions found by the ES and those of any other transpiler.

Test Case #	Test Case	ES	Highest-Fidelity Solution Found
1	Yorktown BV2	88%	89%
2	Yorktown BV3	73%	79%
3	Yorktown BV5	65%	68%
4	Yorktown QFT3	73%	80%
5	Melbourne BV5	59%	62%
6	Yorktown QFT5	34%	50%
7	Melbourne BV8	28%	33%
8	Melbourne QFT3	81%	80%
9	Melbourne BV15	0%	8%
10	Melbourne QFT6	4%	25%

Table 27: ES solution-fidelities versus best found.

The ES only shows signs of convergence on test cases 6 and 7. For these test cases, the ES attains solution-qualities that are on average 24% worse than the best found. This suggests that the ES also settles on poor local optima even when the algorithm shows signs of convergence (albeit, the ES settles on higher-quality local optima than the BRKGA-H). Interestingly, on test case 8 the ES attains a solution of higher-fidelity than any other transpiler, even though there are no signs of convergence by the algorithm for this test case. The provided results indicate that the ES solver needs either more generations with the current configuration or further parameter tuning of μ, κ, and λ.

For several test cases, the best solutions found by both the BRKGA-H and ES are much lower fidelity than the best solutions found after application of the VND. Table 28 compares the fidelities of the best solutions found by the BRKGA-H, BRKGA-H+VND-H, ES, and ES+VND.

	BRKGA-H	BRKGA-H+VND-H	ES	ES+VND
Yorktown BV2	88%	89%	88%	87%
Yorktown BV3	76%	77%	73%	77%
Yorktown BV5	67%	65%	65%	68%
Yorktown QFT3	66%	75%	73%	76%
Melbourne BV5	30%	55%	59%	60%
Yorktown QFT5	18%	41%	34%	40%
Melbourne BV8	5%	27%	28%	32%
Melbourne QFT3	33%	75%	81%	80%
Melbourne BV15	0%	1%	0%	1%
Melbourne QFT6	3%	14%	4%	13%

Table 28: Comparison of fidelities obtained by population-based QLP-solvers and hybrid QLP-solvers.

In 9 out of the 10 test cases, the BRKGA-H+VND-H attains higher-quality solutions than the BRKGA-H. In 8 out of the 10 test cases, the ES+VND attains higher-quality solutions than the ES. Analysis of the data in Table 28 finds that the BRKGA-H+VND-H yielded solutions that are on average 14% better than those of the BRKGA-H. Moreover, the ES+VND yielded solutions that are on average 3% better than those of the ES. Thus, both the BRKGA-H and ES benefit from the VND local search algorithm.

4.3 Computational Effort

In this section, the computational effort for all test cases in Table 17 is provided and analyzed. To collect results, the testing procedure from Section 3.4.3 is used, along with measurements defined in Section 3.4.4. Below, the $transpilation_{time}$ metric for each test case is presented in Figure 34. In addition, Table 29, Table 30, and Table 31 provide the device specifications of all devices used to run the meta-based transpilers.

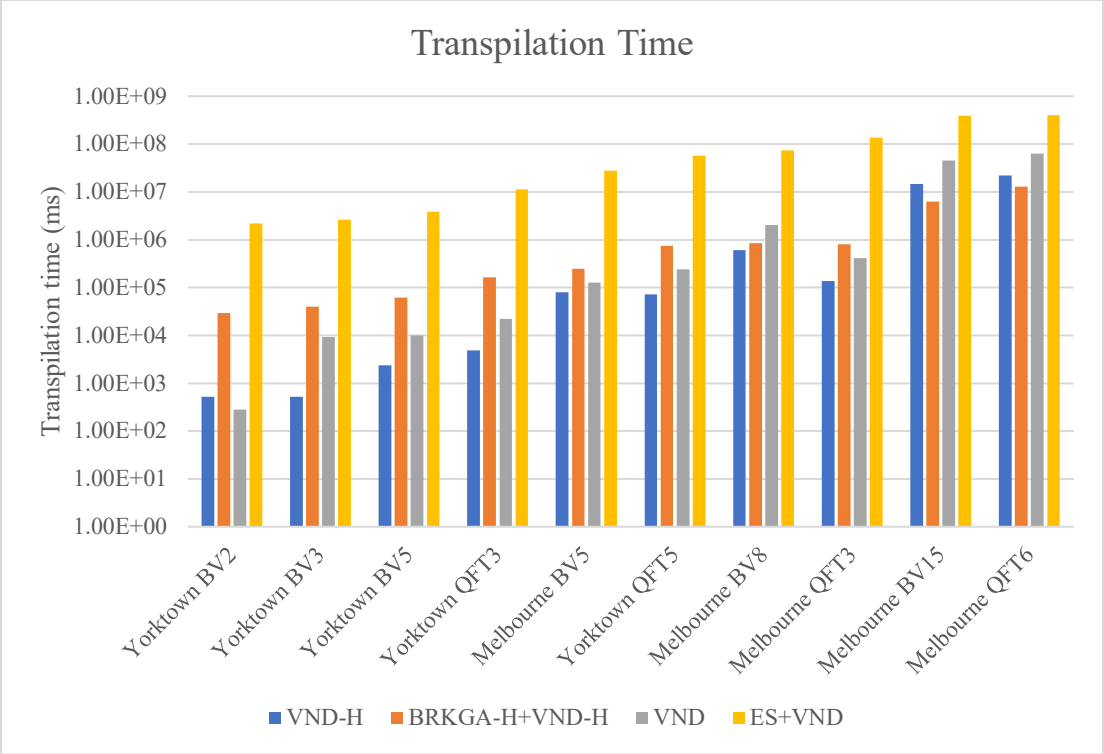


Figure 34: Transpilation times for test cases from Table 17 on various meta-based QLP-solvers.

Specification	Value
Processor	Intel® Core™ i7-8565U CPU @ 1.80GHz 1.99 GHz
Installed RAM	16.0 GB (15.8 usable)
System Type	64-bit operating system, x64-based processor

Table 29: Device specifications for personal computer (PC).

Specification	Value
System Name	mustang.afrl.hpc.mil
Cores/Nodes	48
Core Type/Core Speed	Intel Xeon Platinum 8168/2.7 GHz
Operating System	RHEL

Table 30: Device specifications for Mustang HPC from [45].

Specification	Value
System Name	gaffney.navydsrc.hpc.mil
Cores/Nodes	48
Core Type/Core Speed	Intel Xeon Platinum 8186/2.7 GHz
Operating System	RHEL

Table 31: Device specifications for Gaffney HPC from [46].

First, the BRKGA-H uses only 1 core during execution while the ES uses 48 cores. Next, from Appendix C, the complexities of the true and surrogate objective functions are $O(LN^3)$ and $O(LN^2)$, respectively. Then, for a given population size pop_{size}, number of generations n_{gen}, and number of evolutions n_{evol}, the BRKGA-H and ES execute $n_{evol} \cdot n_{gen} \cdot pop_{size}$ iterations. At each step, the BRKGA-H uses the surrogate objective function on each member of the population and the ES uses the true objective function. For the BRKGA-H, $pop_{size} = L \cdot N$, $n_{gen} = 5000$, and $n_{evol} = 1$. For the ES, $pop_{size} = 10 \cdot L \cdot N$, $n_{gen} = 5000$, and $n_{evol} = 10$.

This means that the BRKGA-H makes $1 \cdot 5000 \cdot L \cdot N = 5000 \cdot LN$ surrogate objective function calls and the ES makes $10 \cdot 5000 \cdot L \cdot N = 500{,}000 \cdot LN$ true objective function calls. Considering the BRKGA-H is executed on a PC via 1 core and the ES is executed on an HPC via 48 cores, the ES makes $\frac{500{,}000 \cdot L \cdot N}{48} \approx 13158 \cdot L \cdot N$ true objective function calls per core. Thus, the ES effectively makes 2.63 times more objective function calls than the BRKGA-H. Moreover, since the BRKGA-H uses the surrogate $f_{obj,u}$ with time complexity $O(LN^2)$ and the ES uses the true objective function f_{obj} with time complexity $O(LN^3)$, each objective function call by the ES is N times longer than the BRKGA-H. Thus, the ES is effectively $2.63 \cdot N$ times slower than the BRKGA-H. The corresponding time complexities of the BRKGA-H and ES are $O(LN \cdot LN^2) = O(L^2N^3)$ and $O(LN \cdot LN^3) = O(L^2N^4)$, respectively. As such, both algorithms are of polynomial time complexity.

The VND, unlike the BRKGA-H and ES, iterates an unknown number of times. At each step, it evaluates all solutions in its current neighborhood until no improving neighbors are found in any of the neighborhoods. All neighborhoods used in the VND are polynomial in size, and therefore each step of the VND is of polynomial time complexity (since the objective functions are of polynomial time complexity). However, there is no sufficient means to bound how many steps the VND will execute. In

Figure 35 and Figure 36, trend lines are calculated to approximate the number of objective function evaluations for a problem-instance of $L \cdot N$ decision variables. The points represent the number of objective function evaluations used by the VND(-H) for each of the QLP problem-instances in Table 17.

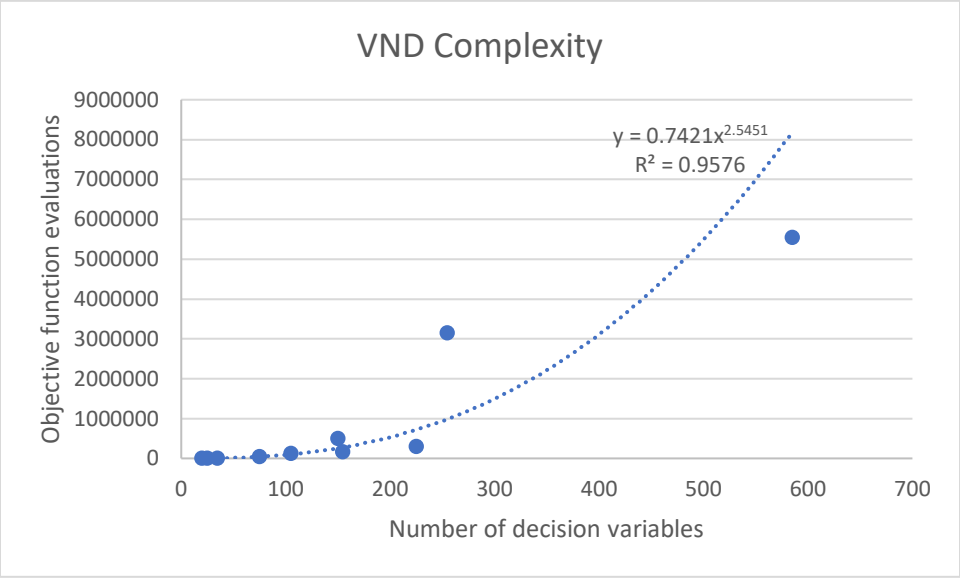


Figure 35: Approximate number of objective function calls by the VND.

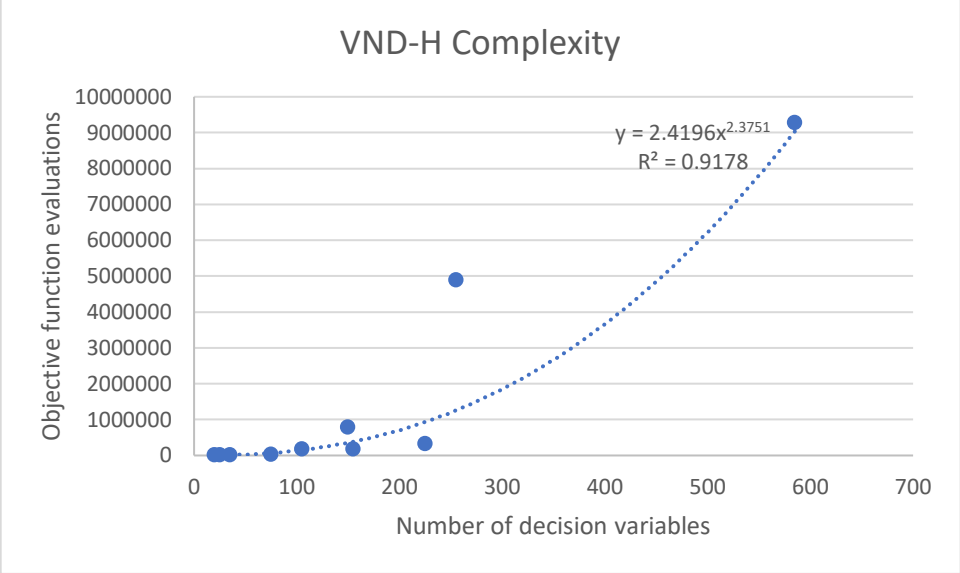


Figure 36: Approximate number of objective function calls by the VND-H.

Figure 35 and Figure 36 suggest that the VND and VND-H make approximately $(LN)^{2.55}$ and $O(LN)^{2.38}$ objective function calls, respectively. Even though the coefficient of determination for both are greater than 0.9 (indicating a good fit), several points are underestimated. For example, consider the points at $x = 150$ and $x = 255$ which represent the Melbourne BV8 and Melbourne BV15 test cases, respectively. Both points are underestimated for the VND and VND-H. Upon closer inspection, the point $x = 105$, which represents the Melbourne BV5, is also underestimated. Since these three test cases are the only BV test cases executed on the IBMQ Melbourne, this suggests that the entanglement requirements of the BV circuits are more challenging to satisfy on the IBMQ Melbourne than those of the QFT. As such, this likely leads both algorithms to prune more solutions in the search space to find improving moves (neighbors).

Next, Table 32 presents the $comp_{effort}$ metric for all meta-based transpilers and test cases.

Test Case #	Test Case	VND-H	BRKGA-H+VND-H	VND	ES+VND
1	Yorktown BV2	0%	0%	0%	3%
2	Yorktown BV3	0%	0%	0%	3%
3	Yorktown BV5	0%	0%	0%	4%
4	Yorktown QFT3	0%	0%	0%	13%
5	Melbourne BV5	0%	0%	0%	32%
6	Yorktown QFT5	0%	1%	0%	67%
7	Melbourne BV8	1%	1%	2%	86%
8	Melbourne QFT3	0%	1%	0%	159%
9	Melbourne BV15	17%	7%	52%	453%
10	Melbourne QFT6	26%	15%	74%	470%

Table 32: Percentage of transpilation window each meta-based transpiler used to find a solution to the QLP for test cases of Table 17.

With the exception of the ES+VND, all meta-based transpilers obtain solutions for all test cases within the calibration data window. Excluding the ES+VND, only the Melbourne BV15 and Melbourne QFT6 take considerably longer to solve for the meta-based QLP-solvers. Furthermore, solutions for test cases 1-8 are obtained within 2% of the calibration window for all meta-based transpilers (again excluding the ES+VND). The ES+VND takes considerably more time because it makes many more true objective function calls than the other meta-based QLP-solvers.

4.4 Fitness Landscapes

The first step of the fitness landscape analysis procedure defined in Section 3.5.3 is to use MDS to plot a small fitness landscape of the QLP. Figure 37 illustrates the fitness landscapes induced by the distance metric defined in Section 3.5.1 coupled with the real and surrogate objective functions for the 1-Bell-state circuit.

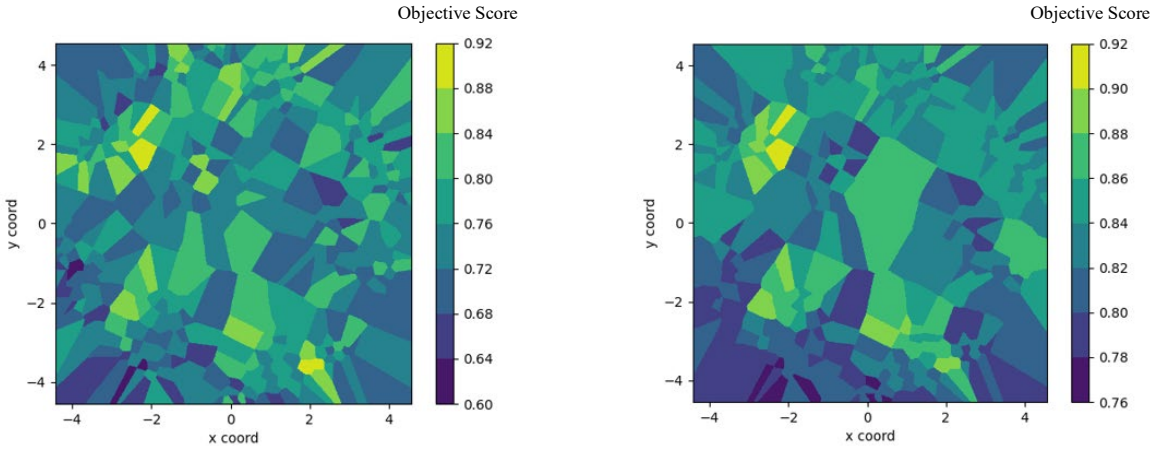


Figure 37: The phenotypic fitness landscapes induced by the distance metric in Section 3.5.1 coupled with the objective function f_{obj} (left) and $f_{obj,u}$ (right).

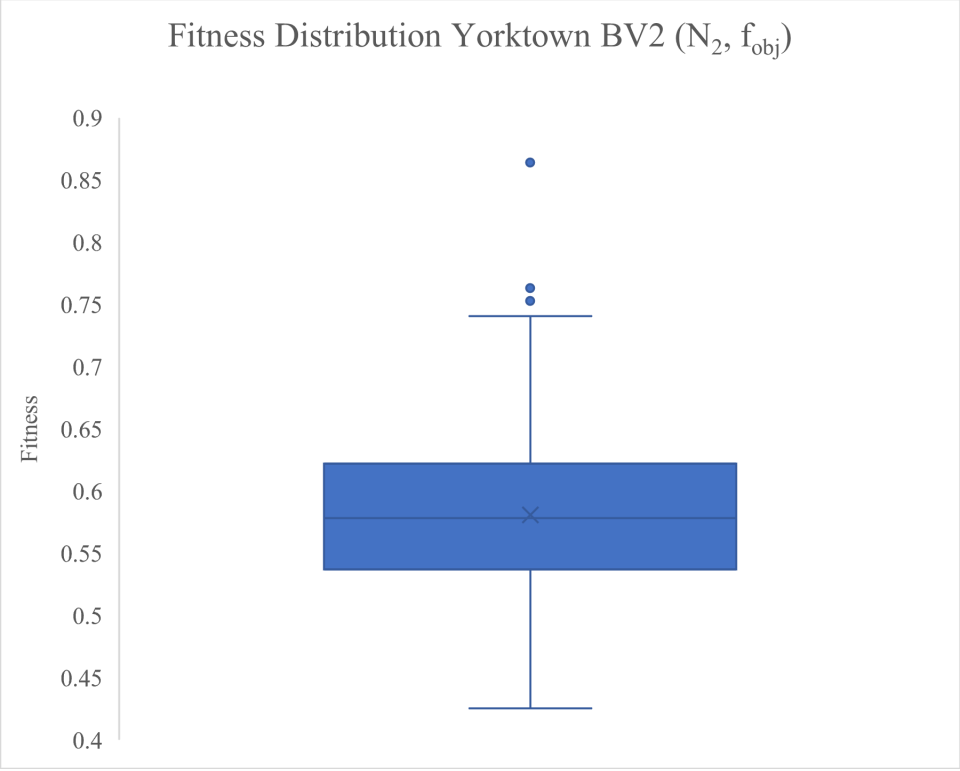
First, notice that both fitness landscapes appear chaotic. In both, there are large regions where the fitness score is nearly the same. Then, on the borders of those regions, either deep basins or higher plateaus are present. There are also multiple globally optimal solutions for this problem-instance (indicated by multiple regions of the highest fitness). Next, notice that the surrogate landscape is flatter than the true fitness landscape. In the surrogate landscape, there are larger regions with similar fitness score, compared to the "pockets" of similar fitness in the true fitness landscape.

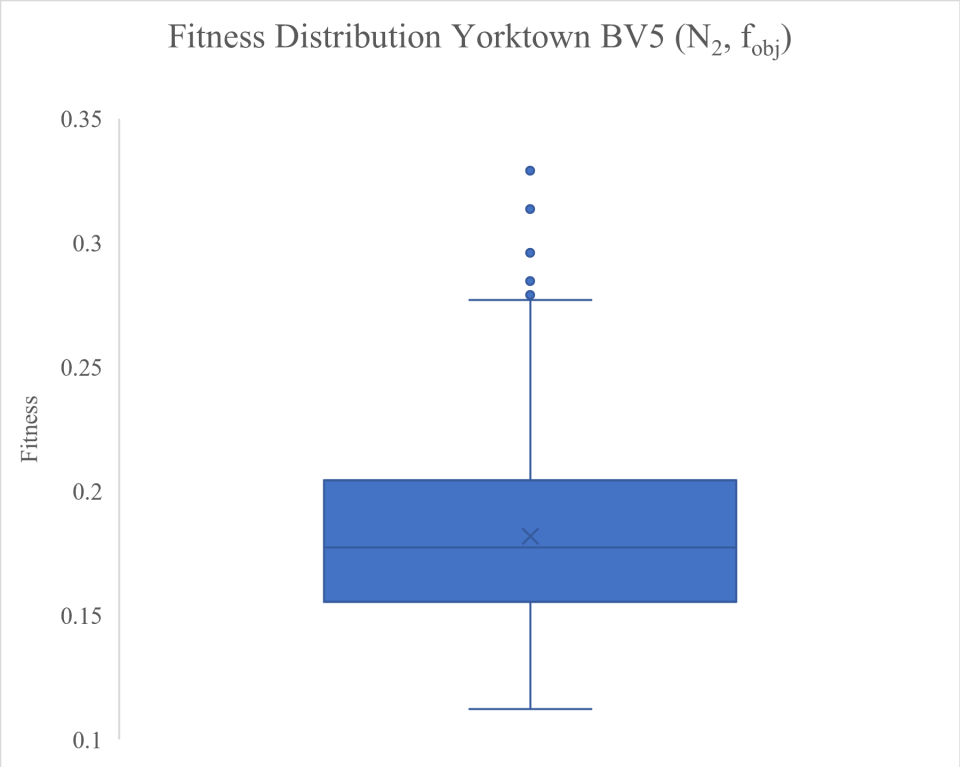
A given point $s_i = (x_i, y_i)$ is some unique permutation of P2L mappings. MDS attempts to place points close to s_i which are small perturbations of s_i (i.e. via the distance metric defined in

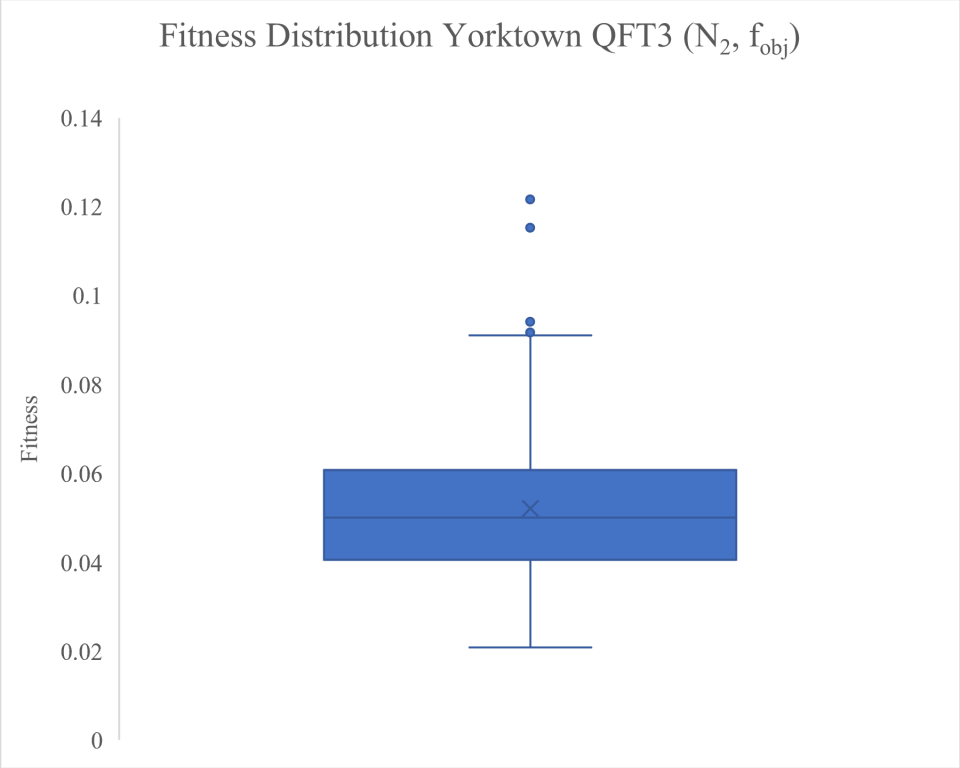
Section 3.5.1). MDS shows that small changes in the representation can largely affect the objective score. This means that the transpiled circuits of two configurations s_i and s_j may be much different from each other, given that there exists pairs of (s_i, s_j) in the landscape which are close to each other but have very different objective scores.

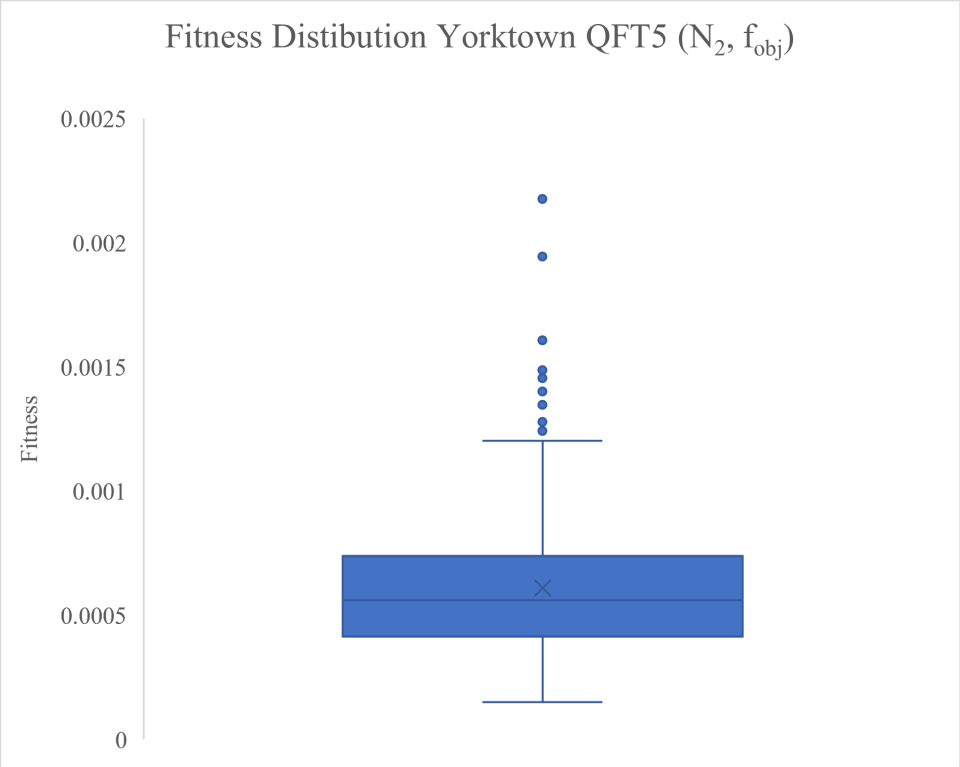
Next, using the techniques specified in Section 3.5.3, the fitness landscapes of the problem-instances provided in Table 19 are analyzed. In this analysis, fitness landscapes are induced by both f_{obj} and $f_{obj,u}$, coupled with neighborhood function N_2. Due to the overall similarity of results for fitness landscapes induced by f_{obj} and $f_{obj,u}$, most results in this chapter only consider f_{obj}.

First, the following four figures show the fitness distribution for the BV2, BV5, QFT3, and QFT5 transpiled to the IBMQ Yorktown. The induced fitness landscapes use neighborhood function N_2 and objective function f_{obj}.

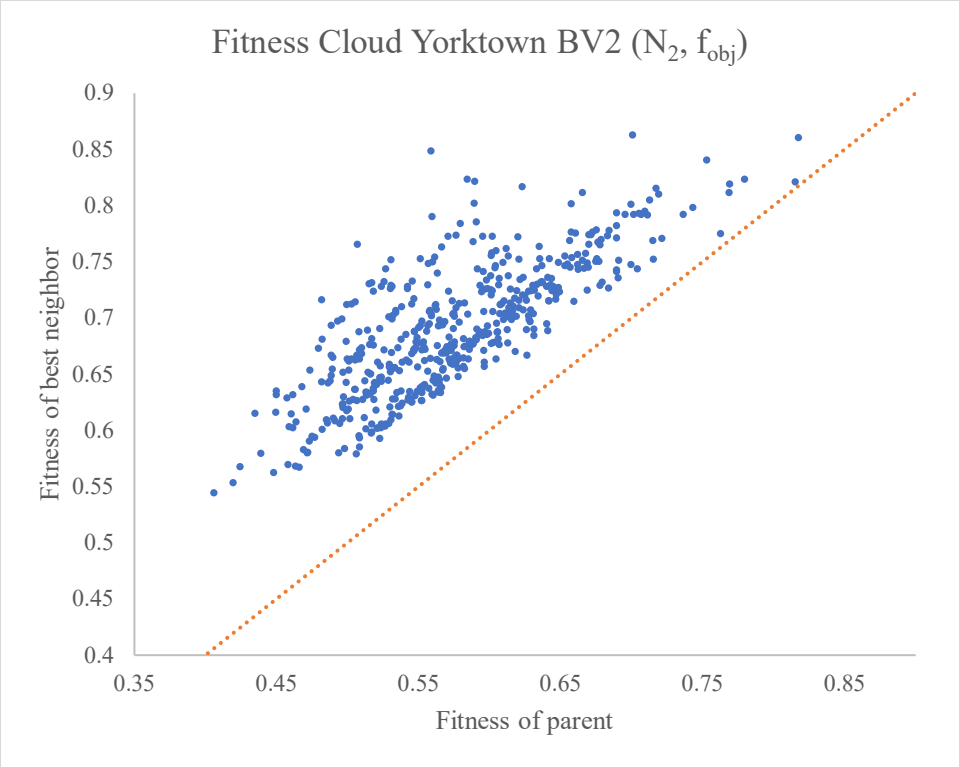


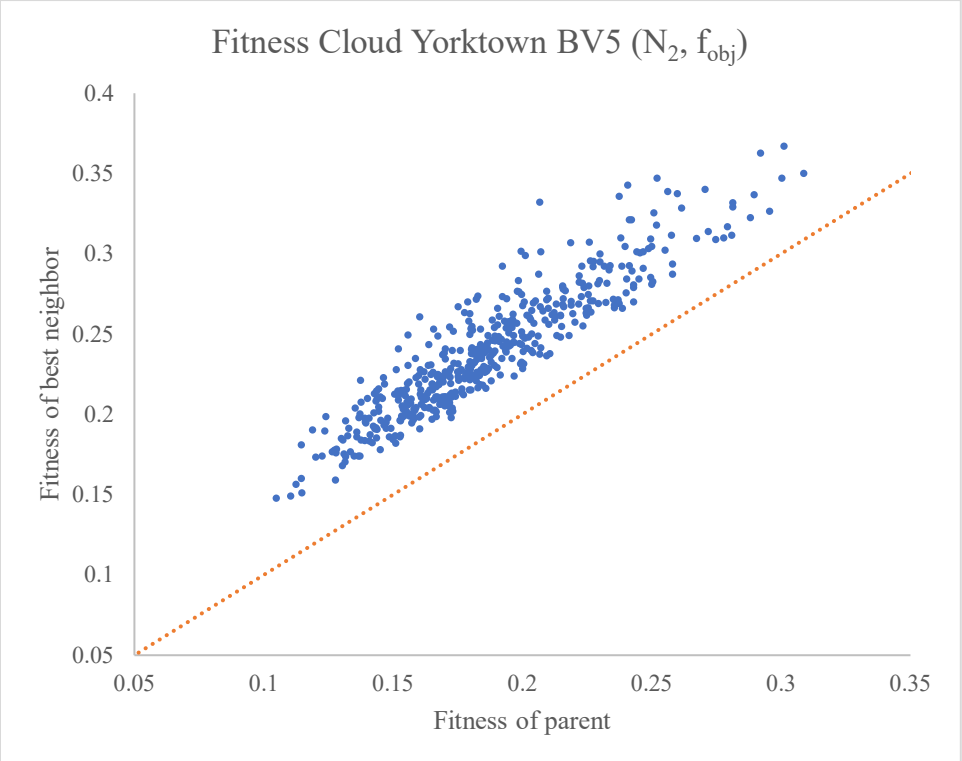


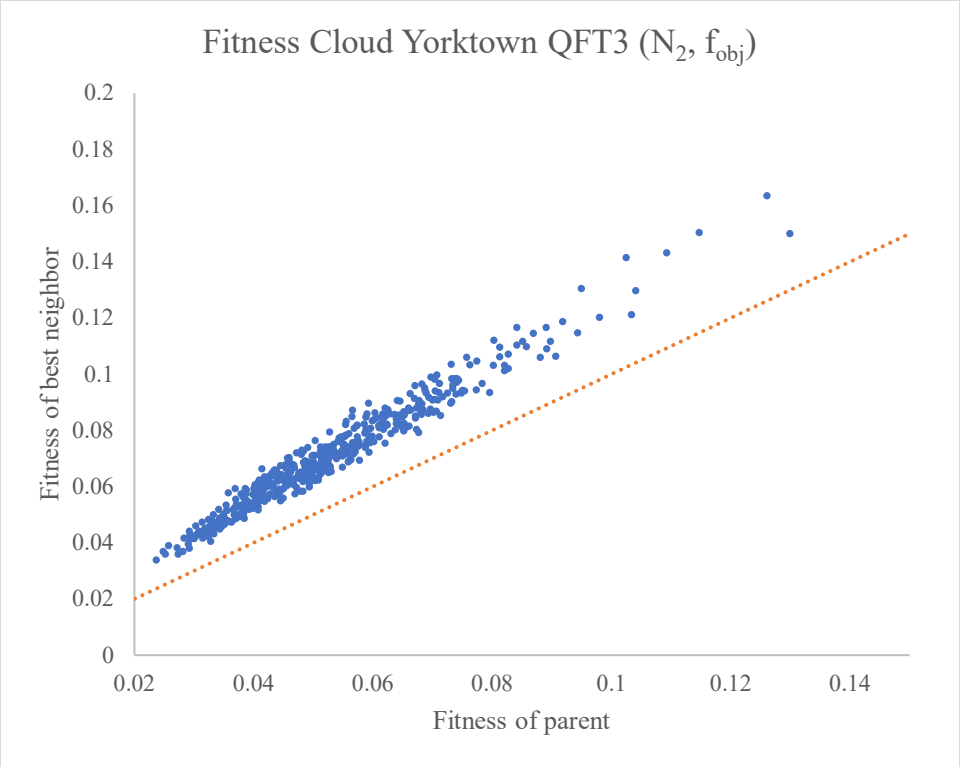


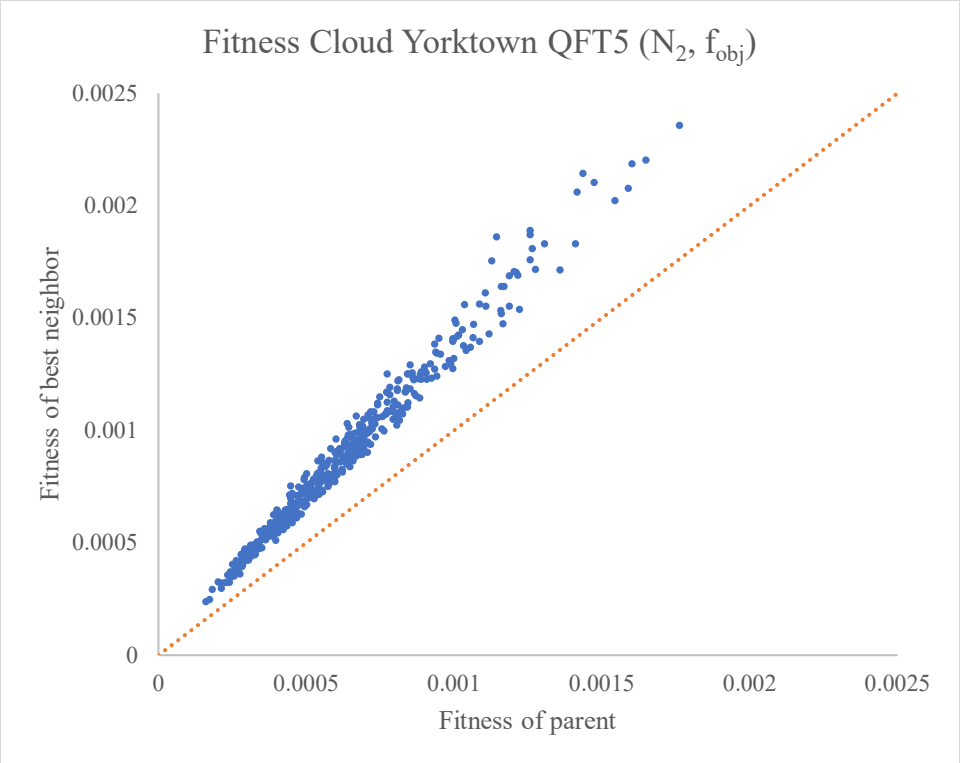


Second, the following four figures show the fitness cloud tests for the BV2, BV5, QFT3, and QFT5 transpiled to the IBMQ Yorktown. The induced fitness landscapes use neighborhood function N_2 and objective function f_{obj}. The dotted line in each figure is the line $y = x$. When points are above that line, this means the fitness of the best neighbor is greater than the fitness of the parent.

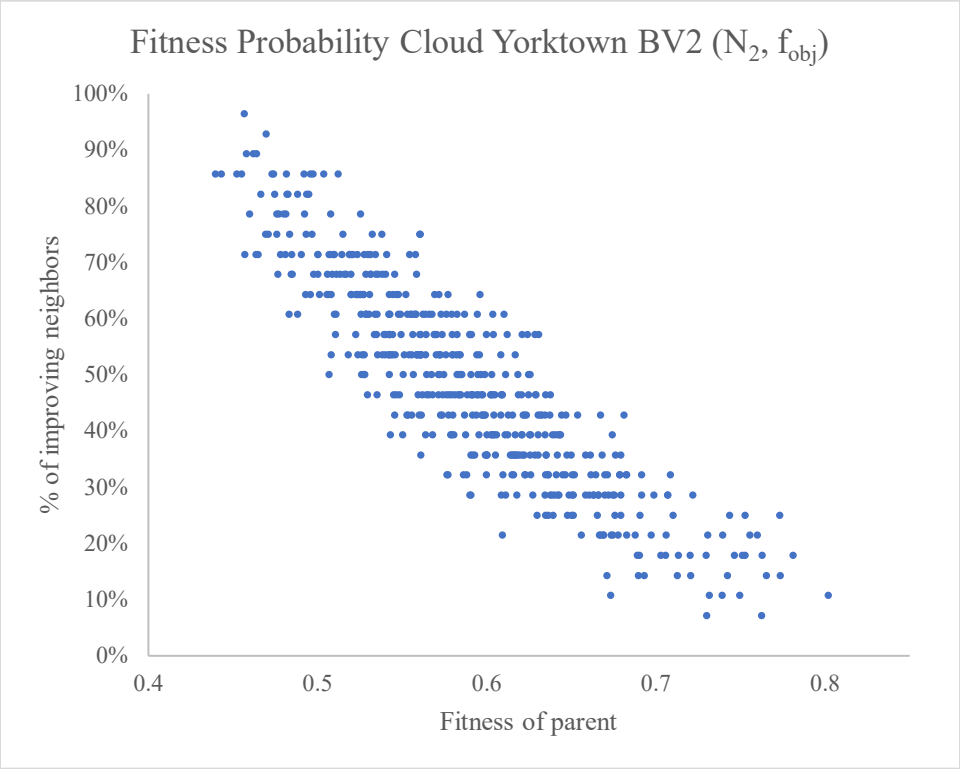


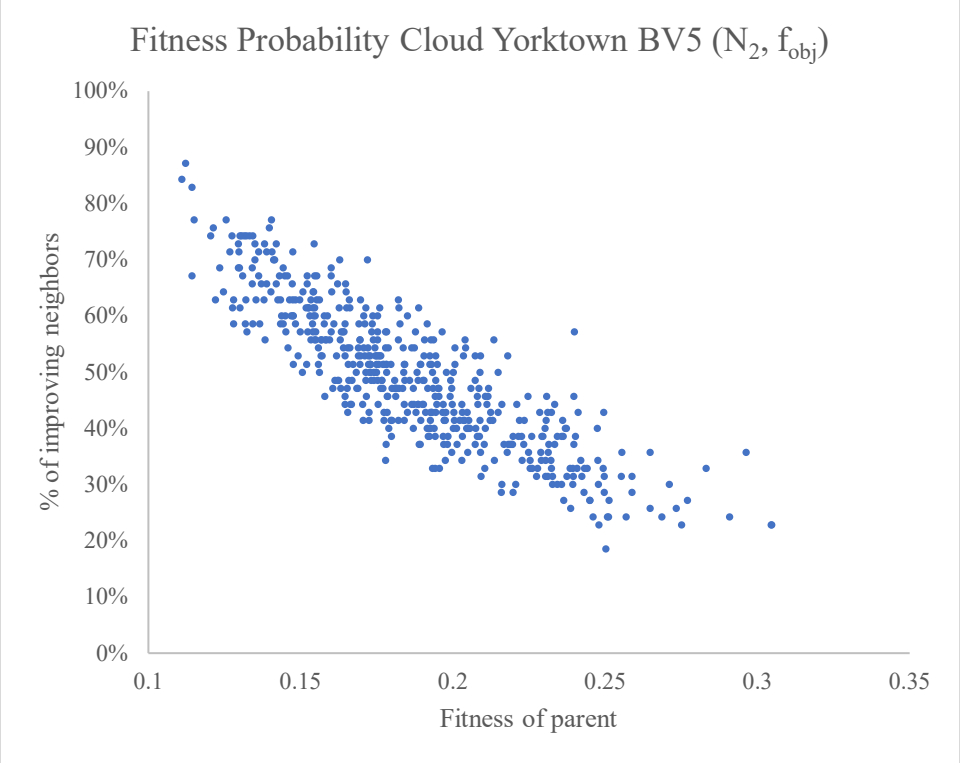


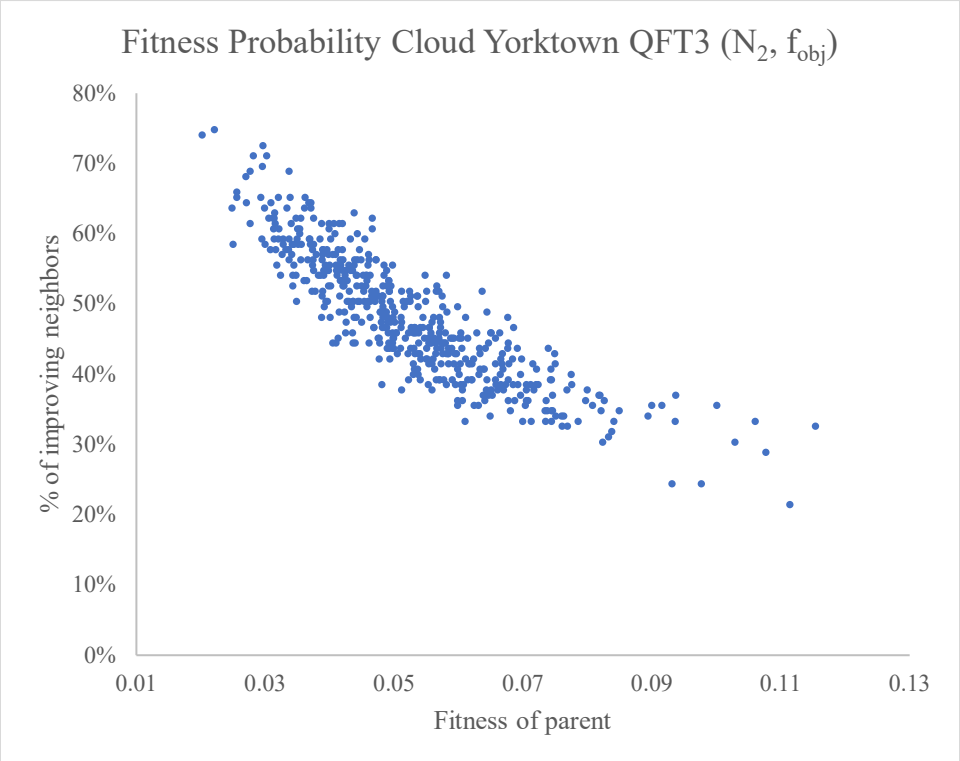


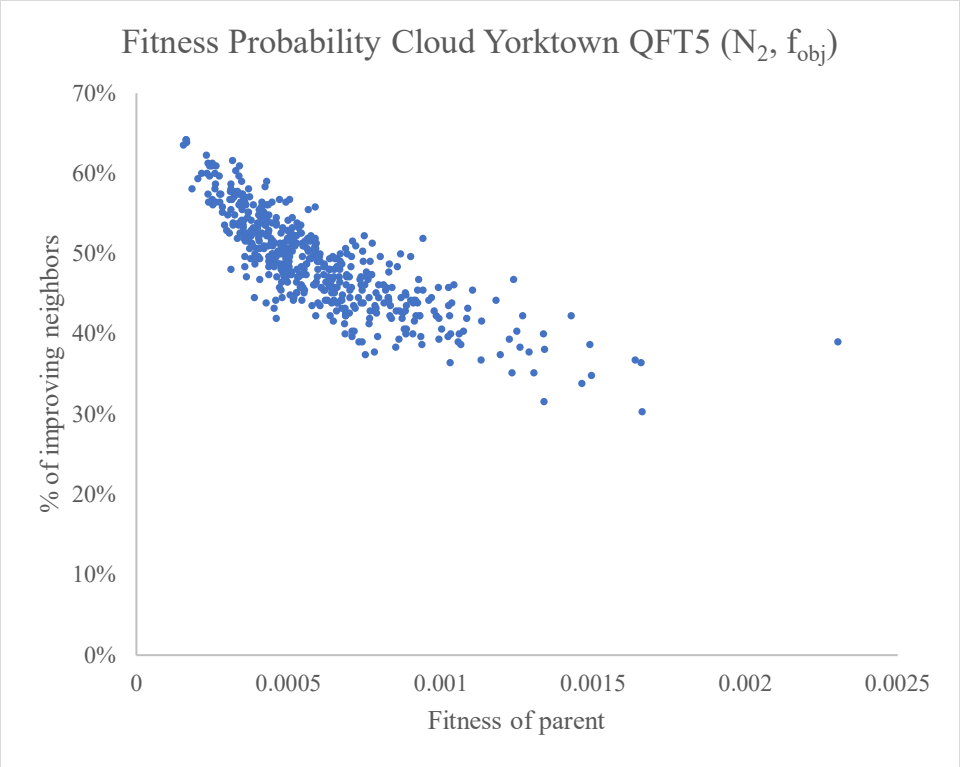


Third, the following four figures show the fitness probability cloud tests for the BV2, BV5, QFT3, and QFT5 transpiled to the IBMQ Yorktown. The induced fitness landscapes use neighborhood function N_2 and objective function f_{obj}.









Test Case	Autocorrelation Value (p)	Escape Probability (e_p)
Yorktown BV2	0.635	0.484
Yorktown BV5	0.841	0.491
Yorktown QFT3	0.915	0.481
Yorktown QFT5	0.963	0.488

Table 33: Autocorrelation and escape probabilities for QLP fitness landscapes induced by N_2 and f_{obj}.

Test Case	Autocorrelation Value (p)	Escape Probability (e_p)
Yorktown BV2	0.747	0.480
Yorktown BV5	0.892	0.472
Yorktown QFT3	0.942	0.426
Yorktown QFT5	0.974	0.439

Table 34: Autocorrelation and escape probabilities for QLP fitness landscapes induced by N_2 and $f_{obj,u}$.

From the fitness landscape analysis of problem-instances above, one of the overarching features prevalent in all is a high autocorrelation value p. When p is close to 0, this indicates that

the landscape is rugged. Conversely, when p is close to 1, this indicates the landscape is flat. Even for the simplest test case, p is 0.635 and monotonically increases to near 1 as the complexity of the test-cases increase. This indicates that the fitness landscape induced by both the surrogate and true objective functions coupled with N_2 is flat. In the analyzed fitness landscapes, search is very difficult. Often, search in flat landscapes like these are called "needle-in-the-haystack" searches.

Next, notice that all analyzed fitness landscapes have an escape probability of around 0.5. This means that progression to higher regions of fitness score in the analyzed landscapes is relatively easy. This is also supported by the fitness cloud (F_c) and fitness-probability cloud (F_{pc}) tests, which show that for a given solution s, the fitness of neighboring solutions of s are higher than that of s in most problem-instances. F_{pc} also shows that as fitness of a given parent solution increases, the percentage of neighbors with higher fitness decreases.

Finally, notice from the fitness distribution (F_d) tests that randomly selected solutions in the search space have similar fitness. Moreover, notice that as problem difficulty increases, the quality of a random solution decreases. This suggests that there exist many low-quality solutions with similar fitness in the search space. Consider Table 35.

Test Case	Number of Decision Variables	Median Sample Objective Score	VND-H Best Objective Score	Median / Best
Yorktown BV2	20	0.578	0.909	0.636
Yorktown BV5	35	0.177	0.651	0.272
Yorktown QFT3	75	0.05	0.753	0.0664
Yorktown QFT5	155	0.00056	0.433	0.00129

Table 35: Median randomly sampled objective score versus best objective score from VND-H.

When the median random sample scores are compared to the best score by the VND-H, the quality of the random sample monotonically decreases as the problem-difficulty increases (based

on number of decision variables). For the easiest test case (Yorktown BV2), the quality of a random sample is about 63% of the best found by the VND-H. In the hardest test case (Yorktown QFT5), the quality of a random sample is about 0.12% of the best found by the VND-H. When considering F_c and F_{pc}, moving from these low-quality solutions to higher-quality solutions is relatively easy (by continuously selecting improving neighbors).

The fitness landscape analysis provides insight as to why local search strategies outperformed population-based metaheuristic algorithms for the QLP. As determined earlier in this section, as the QLP problem-instance complexity increases, the flatness of the fitness landscape increases. For population-based metaheuristics, this means larger perturbations are required to escape the basin of attraction of local optima. During parameter tuning, a mutation rate of 10% is used for the BRKGA. For the ES, κ is set to $L \cdot M$ generations. These configurations may not provide enough genetic diversity to escape local optima once the population settles around a point/set of points in the search space.

In addition, the escape probability for all problem-instances is around 0.5. This is helpful for local search algorithms because for a given point in the search space, there is a high probability that a neighboring point has higher fitness. The fitness landscape analysis only analyzed N_2, but the research results suggest the reason the VND works well for the QLP is because once no more improving neighbors are found in N_2, the lower locality neighborhoods can create large enough perturbations to escape the basin of attraction. Moreover, these lower locality neighborhoods use problem-domain specific knowledge to guide the search process towards solutions that coincide with the general structure of highly-fit optima (i.e. high-fidelity P2L mappings for each layer of the circuit where the aggregate distance between all pairs of consecutive layouts is minimal).

4.5 Summary

In Sections 4.2 and 4.3, the quality of solutions and computational effort are presented, respectively. Then, Section 4.4 provides the results after fitness landscape analysis of the QLP. All three sections seek to address the research questions posed in Chapter I. Section 4.2 and 4.3 provide results that address the effectiveness and efficiency of various meta-based QLP-solvers. Finally, Section 4.4 provides results to describe various fitness landscapes of the QLP.

V. Conclusions

5.1 Overview

In Section 5.2, the research questions posed in Chapter I are revisited. Section 5.3 provides the contributions of this work to the quantum computing research field. In Section 5.4, future work in QLP-solver development is provided based on insights made during this research effort. Finally, Section 5.5 provides closing remarks to summarize the work in this study.

5.2 Addressing Research Questions

At this point, the research questions posed in Chapter I are revisited and addressed.

1. How effective and efficient are various metaheuristic algorithms at finding high-quality solutions to the QLP?
2. For various QLP problem-instances, how can the topology of the fitness landscape induced by the representation, objective function, and search operator(s) devised for the metaheuristics based QLP-solvers be characterized?

For research question #1, metaheuristic algorithms appear to be a fruitful approach to finding high-quality solutions to the QLP. Meta-based transpilers obtained the same or higher-quality solutions in 7 of the 10 test cases. Moreover, the VND-H had average solution-quality gains of 19%, 7%, and 5% over the three benchmark transpilers. However, no meta-based transpiler outperformed the benchmark transpilers on the two hardest test cases. This suggests that while the meta-based transpilers may be effective for smaller-sized problem instances of the QLP, they may not be effective for larger-sized problem instances.

Next, an argument could be made that the poor runtimes shown in Chapter IV exhibit that metaheuristic approaches are impractical. However, both the objective functions and metaheuristic algorithms run in polynomial time (for the VND, this was only observed). Moreover, the task of evaluating an objective function on a collection of candidate solutions is an

embarrassingly parallel task for which HPCs can evaluate much more efficiently. For all meta-based QLP-solvers, evaluating a collection of solutions at each step is the most computationally-intensive task. Further, few enhancements are done in this research effort to optimize the objective functions. Most of the objective function software is written in the Python programming language. If instead the objective function software is written in a compiled language, such as C++, performance could be enhanced by orders of magnitude.

For research question #2, the most defining features observed from fitness landscape analysis of the induced fitness landscapes are:

1. The fitness landscape of the QLP is relatively flat, as shown by the autocorrelation function tests. In addition, the MDS plots further support the flatness of the QLP landscape and show that there are numerous plateaus in the QLP landscape.
2. While the landscape is relatively flat, the escape probability of around 50% for all problem-instances suggests that there is a slight incline to the plateaus. However, the direction of that incline is not guaranteed to lead towards a global optimum (otherwise local search with N_2 would always yield a highly-fit optimum, which was not observed).
3. There are many low-quality solutions in the QLP solution space. When randomly sampling, the fitness scores of the random samples are not close to the score of highly fit optima.

The observed characteristics of the QLP fitness landscape suggest that traversing the high-dimensional space of the QLP is a difficult task. Moreover, they illustrate why the VND (with both the true and surrogate objective functions) was an effective metaheuristic algorithm to employ for the QLP. Talbi suggests a common technique to "break plateaus" is to change the objective function [6]. In the VND algorithm with $\gamma = \text{True}$ (the algorithm employed by the VND-H), first a round of VND is carried out with the surrogate objective function $f_{obj,u}$. Then, when no improving solutions are found with that objective function, the objective function is

changed to f_{obj}. Thus, the VND with $\gamma = \text{True}$ employs both objective space change and neighborhood function change, which likely helps guide the search process to better local optima than other metaheuristic algorithms for the QLP.

5.3 Contribution

In this research effort, extensions of the mathematical model of the QLP are devised. In previous research, the QLP is defined as a constrained optimization problem with nonlinear constraints. In this work, the QLP is formulated as a single-objective constrained optimization problem with linear constraints via problem reductions made possible by the GBO and TokSP solver. Next, extensions of objective functions devised by previous research efforts are created to mathematically formalize and objectively score solutions to the QLP. The devised representation and objective functions may provide future researchers with a framework to develop novel optimization algorithms to find high-quality solutions to the QLP.

Next, the mathematical model of the QLP is integrated into various metaheuristic algorithm domains. Specifically, a single-solution (VND) and two population-based (BRKGA and ES) metaheuristic algorithms are devised to find solutions to the QLP. While previous research efforts mainly use local search strategies to find solutions to the QLP, none specifically use a VND algorithm as done in this work. In addition, the BRKGA and ES-based QLP-solvers lay a foundation for population-based QLP-solvers, an area of research that has not been extensively considered in previous research efforts.

Finally, the fitness landscape of several QLP problem-instances is analyzed. The analysis of the induced QLP fitness landscapes provides insight into global and local characteristics of the QLP fitness landscape. Empirical analysis of QLP fitness landscapes is not conducted in previous research efforts. The observed characteristics of QLP fitness landscapes in this research provide problem-domain specific knowledge to the QLP-solver development community that can be used to tailor (meta)heuristics-based QLP-solvers in future research endeavors.

5.4 Future Work

First, the meta-based QLP-solvers rely on a TokSP solver to calculate minimal sequences of SWAP operations to permute between layouts. In this research, a TokSP solver is implemented based on a high-level algorithm from Miltzow et al. [35]. In their paper, they claim that their TokSP algorithm is of polynomial time complexity and is a 4-approximation algorithm. In this work, no optimizations are introduced for the provided TokSP algorithm. Future research efforts can devise and implement an optimized TokSP solver, as well as characterize the best-, average-, and worst-case time complexities of that solver. In this work, surrogate objective functions are created to address the poor asymptotic execution time of the TokSP solver. Potentially, the need for surrogate objective functions could be eliminated given an optimized TokSP solver.

In addition, the TokSP algorithm provided by Miltzow et al. considers the edge weights to be 1 for all vertices of the graph. For the QLP, the edge weights are instead probabilities of successfully swapping along an edge with edge-weights in $[0,1]$. As such, there is a need for a TokSP solver that finds a fidelity-optimal sequence of SWAPs to permute between two given configurations. The author's review of the literature did not identify research that has provided such an algorithm (let alone with some approximation guarantee).

Next, due to the poor asymptotic execution time of the TokSP-Solver, lower and upper bound surrogate objective functions are devised. While these surrogates provide a useful approximation of the true objective function, the bounds are not tight. For future research, development of a surrogate that more closely mimics the true objective function would likely guide the search process towards highly fit optima in absence of the true objective function. Moreover, the true objective function appears, from Chapter IV, to not provide an amenable fitness landscape for a search process to optimize. A clever surrogate objective function may induce a more amenable fitness landscape for which a search process can navigate to the "needles" in the "haystack" of the search space (as illustrated in Section 4.4, where the fitness landscape tends towards flatness as

problem difficulty increases). Following up on suggestions for a fidelity optimal TokSP-Solver, a clever surrogate fitness function would, for example, approximate the lower bound fidelity of swapping between two configurations of tokens on a graph.

The objective functions devised for the QLP only consider gate fidelities. A fruitful direction for future research efforts is to incorporate T_1 and T_2 times into the objective function as well. As T_2 time is generally less than T_1 time, the incorporation of T_2 time is sufficient to provide a more meaningful objective score for a given solution. The T_2 decay can be characterized by $0.5 + \left(0.5 \cdot e^{\frac{-t}{T_2}}\right)$, where t is the amount of time since the qubit has been placed in the state $|+\rangle$, and T_2 is the decay constant associated with spin-relaxation (for a given qubit).

Based on the results gathered in Chapter IV, the VND-based QLP-solver appears to be the backbone of all devised meta-based QLP-solvers. The results of this research effort show that the population-based QLP-solvers, when not augmented with the VND, perform poorly. The VND, however, uses a variety of neighborhood functions to escape the basin of attraction of local optima that the population-based metaheuristic algorithms tend to get stuck in. To accomplish this, weaker locality neighborhoods that transform a given candidate solution towards the anticipated global optimal structure are evaluated when stronger locality neighborhoods cannot perturb the candidate in a way that increases fitness. Future research efforts could focus on evaluating the effectiveness of specific neighborhoods and devising new neighborhood functions that create more useful perturbations that lead towards global optima. As the VND currently operates, much time is wasted evaluating neighborhoods that do not contain improving neighbors. Moreover, many neighbors are selected regardless of how much they improve fitness. Future research efforts could alter acceptance criterion to only accept solutions that improve the current candidate's fitness by some variable ϵ to enhance performance of the VND.

Another option for future work is to use metaheuristic algorithms to find high-quality solutions to a sub-problem of the QLP, such as finding an initial mapping. For example, Liu et al. use a simulated-annealing algorithm to find solutions to the initial mapping problem of the QLP. Other metaheuristic algorithms can be explored to tackle one of the subproblems of the QLP. Solving subproblems, rather than the entire QLP, would significantly reduce the number of decision variables even for larger problem-instances. By reducing the size of the search space, metaheuristic algorithms may yield higher-quality optima. Moreover, a cleverer representation, objective function, and search operator may induce a fitness landscapes for the subproblem that are easier for a search process to traverse than the induced fitness landscapes in this research effort.

Finally, the population-based QLP-solvers tend to get stuck in poor local optima. One way future efforts could address this issue is to tune the hyperparameters of the BRKGA and ES more intelligently. Hyperparameters are configured based on previous research efforts and experimentation, but much more analysis can be done to find parameter tunings which lead the algorithms to convergence on higher-quality optima (e.g., higher mutation rate, lower elitism). In addition, the population-based QLP-solvers can be augmented with a local search strategy. Specifically, one or several neighborhood functions can be used to perturb k solutions in the current population. For each of the k selected members, the local search strategy attempts to find an improving neighbor. Then, if one is found, the member is updated to its improving neighbor. This variant of hybrid metaheuristic algorithms is described in detail by Talbi [6], where he describes combining local search with population-based metaheuristics. Based on the effectiveness of local search strategies for the QLP, augmentation of the population-based solvers with iterative local search may yield faster convergence on higher-quality optima.

5.5 Concluding Remarks

The potential for meta-based QLP-solvers to find higher-fidelity solutions to the QLP appears feasible from results gathered in this research effort. While there is a runtime tradeoff between the SOTA and meta-based QLP-solvers, certain situations require the highest possible fidelity transpiled circuit, where researchers are willing to sacrifice runtime for quality of solutions. Talbi describes two extreme types of problems: design problems and control problems. Design problems are generally solved once, whereas control problems must be solved frequently. The QLP lies somewhere between these two extrema and is considered a "medium-term problem" [6]. Calibration data is updated approximately every 24 hours, which provides a window of time for optimization algorithms to prune for high-quality solutions to the QLP.

The fitness landscape analysis provides insight into both global and local characteristics of the induced fitness landscape for the QLP. Globally, the QLP fitness landscape appears flat, with numerous plateaus. Locally, the fitness landscape has a small gradient for local search operators to follow (but the direction of improvement is not guaranteed to lead towards global optima). Moreover, most solutions in the search space are low-quality. With this knowledge, future researchers can tailor optimization algorithms to traverse the observed landscape. Overall, the fitness landscape analysis shows that the QLP is indeed a difficult problem to optimize.

The implementation of a TokSP-Solver, novel linearly-constrained mathematical model of the QLP, and GBO offer QLP-researchers with additional tools that can be used in future research endeavors. In addition, the fitness landscape analysis procedure provides QLP-researchers with an empirical approach to characterize the topology of their induced fitness landscape. In this research effort, the QLP is first integrated in various metaheuristic algorithm domains. Steps are then taken to integrate a mathematical formulation of the QLP into various metaheuristic algorithm domains. The results in Chapter IV indicate that single-solution based metaheuristics may be particularly applicable to the QLP. Thus, future research endeavors should strongly

consider the design decision to sacrifice runtime for quality of solutions as done in this body of work, as doing so may permit complex circuits to yield the correct solution(s) in the NISQ-era of quantum computing.

Appendix A. Number of CNOT Gates Used in Generalized Bridge Operation

This section addresses the number of C_{NOT} operations performed in the GBO algorithm between pairwise adjacent vertices of path p, where p is the shortest weighted path between distant control and target qubits p_0 and p_d, separated by distance d. Consider the GBO algorithm from chapter III.

Algorithm Generalized Bridge Operation.

Preliminaries: Let q_c and q_t be the control and target logical qubits of a C_{NOT} operation, respectively. Let P_γ and P_τ be the physical qubits that hold q_c and q_t, respectively. Let p be a sequence holding the shortest weighted path between P_γ and P_τ in G (where the edge weights are the 2-qubit error rates between adjacent physical qubits. The elements of p are unique physical qubits from $V(G)$. Further, $p_0 = P_\gamma$ and $p_d = P_\tau$.). Let $d = |p| - 1$ (where $|p|$ denotes the number of vertices in p).

C_{NOTs} = [] ; /* List to hold C_{NOT} ops to perform, defined by (P_i, P_j) pairs where P_i is the control and P_j is the target */

/* Phase ρ_1 : descending C_{NOT} operations */
/* "Descending" refers to moving backwards along path p */
For ($i = d$; $i > 0$; i--) **Do**
  $C_{NOTs}.append((p_{i-1},\ p_i))$;
End For

/* Phase ρ_2 : ascending C_{NOT} operations */
/* "Ascending" refers to moving forwards along path p */
For ($i = 1$; $i < d$; i++) **Do**
  $C_{NOTs}.append((p_i,\ p_{i+1}))$;
End For

/* Phase ρ_3 : repair */
For ($i = 1$; $i \leq (2 \cdot d) - 3$; i++) **Do**
  $C_{NOTs}.append(C_{NOTs}[i])$;
End For

Output C_{NOTs}, which is a sequence of (P_i, P_j) pairs which must be performed, in order, to simulate $C_{NOT}(P_c, P_t)$.

Phases ρ_1 and ρ_2 each perform one C_{NOT} between control p_{d-1} and p_d. Note that phase ρ_3 never inserts a C_{NOT} between p_{d-1} and p_d since ρ_3 only performs the C_{NOT} operations that got $p_0 \ldots p_{d-1}$ entangled with p_0. Thus, a total of 2 C_{NOT}s are performed by the GBO algorithm between p_{d-1} and p_d.

Phases ρ_1 and ρ_3 each perform one C_{NOT} between p_0 and p_1. Note that phase ρ_2 never inserts a C_{NOT} between p_0 and p_1. Thus, a total of 2 C_{NOT}s are performed by the GBO algorithm between p_0 and p_1.

Phases ρ_1 and ρ_2 each perform one C_{NOT} between all intermediate pairwise adjacent qubits in p; that is, $\big((p_1, p_2), (p_2, p_3), \ldots, (p_{d-2}, p_{d-1})\big)$. In phase ρ_3, these same two C_{NOT}s are applied again to revert all intermediate qubits back to their original states. Thus, a total of 4 C_{NOT}s are performed between all intermediate pairwise adjacent qubits of p.

Appendix B. Token-Swapping Problem Algorithm

Miltzow et al. present a 4-approximation algorithm for the TokSP on general graphs [35]. To devise the pseudocode below, the algorithmic steps described by Tilman et al. are used. The following pseudocode presents an implementable algorithm for a TokSP-Solver. To the best of my knowledge, this is the first presentation of an implementable algorithm for Miltzow et al.'s approximation algorithm. In other words, the high-level algorithmic steps of Miltzow et al.'s TokSP algorithm are refined to a form that is easier to implement for computer scientists.

In the provided TokSP algorithm, logical qubits are synonymous with tokens, physical qubits are the vertices of graph G, and λ_i and λ_f are P2L mappings (token placements). Furthermore, this algorithm works even when the number of logical qubits (tokens) is less than the number of physical qubits (vertices in G). Let q_ϵ denote an unused logical qubit (token).

Algorithm For Token-Swapping problem.

Inputs: A graph G, an initial configuration λ_i, a final configuration λ_f, and an optional random seed r.

If $r \neq \emptyset$ **Then** initialize pseudorandom number generator with r ;

$swap\_seq$ = [] ; /* list to hold generated SWAP sequence */

$\lambda_p = \lambda_i$; /* partial solution */

While $\lambda_p \neq \lambda_f$ **Do** /* loop 1 */
  /* define directed graph F on $V(G)$ */
  $F = Graph()$; $F.add\_nodes\_from(V(G))$;

  For P_v, P_w in $E(G)$ **Do** /* loop 2 */
    /* simulate swapping over edge (P_v, P_w) */
    $\lambda_{temp} = \lambda_p$;
    $\lambda_{temp}(P_v), \lambda_{temp}(P_w) = \lambda_{temp}(P_w), \lambda_{temp}(P_v)$;

    $q_v = \lambda_p(P_v)$; /* get token assigned to P_v */
    If $q_v \neq q_\epsilon$ **Then**
      /* get distance of q_v from source to target before and after swapping using $d()$ from section 3.3.1.4 */
      $dist\_before = d\left(\lambda_p^{-1}(q_v), \lambda_f^{-1}(q_v)\right)$;

$dist\_after = d\left(\lambda_{temp}^{-1}(q_v), \lambda_f^{-1}(q_v)\right)$;

/* if the token on P_v reduces its distance to its target by swapping along current edge, add directed edge to F */
If $dist\_after < dist\_before$ **Then**
$F.add\_edge\left((P_v, P_w)\right)$;
End For /* loop 2 */

/* choose any vertex that does not hold the right token */
$verts\_not\_holding\_right\_tokens$ = [] ;
For P_i in $V(F)$ **Do** /* loop 3 */
If $\lambda_f(P_i) \neq \lambda_p(P_i)$ and $\lambda_p(P_i) \neq q_\epsilon$ **Then**
$verts\_not\_holding\_right\_tokens.append(P_i)$;
End For /* loop 3 */
$v = random.choice(verts\_not\_holding\_right\_tokens)$;

/* construct a directed path from v by following directed edges of F */
$v_p = v$;
$done$ = False;
$path = [v_p]$; /* list to hold path explored */

/* loop until a happy SWAP (i.e. a cycle in path) found or an unhappy SWAP (i.e. vertex with an out-degree of 0) encountered */
While not $done$ **Do** /* loop 4 */
$v_p = random.choice\left(F.neighbors(v_p)\right)$; /* choose any neighboring vertex of v_p */
$path.append(v_p)$;

If $path.count(v_p) > 1$ **Then**
$vp\_first\_idx = path.index(v_p)$; /* get path index of first occurrence of v_p */
$vp\_last\_idx = |path| - 1$; /* last index of the path is the second occurrence of v_p */

$path = path[vp\_first\_idx : (vp\_last\_idx - 1)]$; /* extract cycle */
/* the cycle is broken by omitting the last element */

$path.reverse()$; /* reverse path to perform SWAPs in correct order */

/* perform happy SWAP chain */
For ($i = 1$; $i < path.size()$; i++) **Do**
$P_i, P_j = path[i-1], path[i]$;
$\lambda_p(P_i), \lambda_p(P_j) = \lambda_p(P_j), \lambda_p(P_i)$;
$swap\_seq.append((P_i, P_j))$;
$done = True$;

Else If $F.out\_degree(v_p) = 0$ **Then**
/* extract last two elements of path */
$p\_len = |path|$; /* get length of path */
$P_i, P_j = path[p\_len - 2], path[p\_len - 1]$;

/* perform unhappy SWAP */
$\lambda_p(P_i), \lambda_p(P_j) = \lambda_p(P_j), \lambda_p(P_i)$;
$swap\_seq.append((P_i, P_j))$;
$done = True$;
End While /* loop 4 */

End While /* loop 1 */

Output $swap\_seq$, which is a sequence of (P_i, P_j) pairs that when used as transpositions applied to λ_i (in order), permute configuration λ_i to configuration λ_f.

Time Complexity: Since at each iteration of loop 1 either a happy or unhappy SWAP is performed (otherwise, all tokens have arrived at the targets and the algorithm is done), and [35] claims their algorithm is a 4-approximation and the upper bound number of SWAPs required is $4 \cdot D$, the outermost while loop executes at most $4 \cdot D$ times, where D is the aggregate distance of all tokens to from their positions in λ_i to their destinations in λ_f. Then, loop 2 executes E times, where $E = |E(G)|$. Next, loop 3 executes at most E times since in the worst case all tokens are not at their target vertices. Finally, loop 4 executes at most N times, where $N = |V(G)|$. This occurs when the entire graph is a happy SWAP cycle. Thus, the overall time complexity of the provided TokSP algorithm is $O\big(4D \cdot (E + E + N)\big) = O\big(4D \cdot (2E + N)\big)$. From Miltzow et al. [35], D is bounded by N^2 for general graphs. Thus, the time complexity is $O\big(4N^2 \cdot (2E + N)\big) \rightarrow O\big(N^2 \cdot (E + N)\big)$. Considering $E \approx N$ for NISQ-era QCs, the time complexity for TokSP-instances for NISQ-era QCs can be expressed as approximately $O\big(N^2 \cdot (N + N)\big) \rightarrow O(N^3)$.

Appendix C. Time Complexity of Objective Functions

In this section, the worst-case time complexity of both the surrogate ($f_{obj,u}$) and true objective functions (f_{obj}) from Section 3.3.1 are presented. Table 36 and Table 37 present the worst-case time complexity derivations for f_{obj} and $f_{obj,u}$, respectively.

Step #	Step	Description
1	$f_{obj} = O(f_q) + O(f_s)$	f_{obj} is composed of two separate operations: f_q and f_s.
2	$\rightarrow O(f_{1q}) + O(f_{2q}) + O(f_s)$	Expanding f_q.
3	$\rightarrow O\left(\prod_{\lambda_i \in s}\prod_{q_j \in T} s_1(\lambda_i, q_j)\right) + O\left(\prod_{\lambda_i \in s}\prod_{q_j \in T}\prod_{q_k \in T} s_2(\lambda_i, q_j, q_k)\right) + O(f_s)$	Expanding f_{1q} and f_{2q}.
4	$\rightarrow O(LN) + O(LN^2) + O(f_s)$	Given that $\|T\| = N$, $\|s\| = L$, and both s_1 and s_2 are $O(1)$ operations, $O(f_{1q}) = O(LN)$ and $O(f_{2q}) = O(LN^2)$.
5	$\rightarrow O(LN) + O(LN^2) + O\left(\prod_{\lambda_i, \lambda_{i+1} \in s} c(w(\lambda_i, \lambda_{i+1}))\right)$	Expanding f_s.
6	$\rightarrow O(LN) + O(LN^2) + O\left(\prod_{\lambda_i, \lambda_{i+1}} O(N^3)\right)$ $\rightarrow O(LN + LN^2 + O((L-1) \cdot O(N^3))$ $\rightarrow O(LN) + O(LN^2) + O(LN^3)$	Considering $\pi_s = w(\lambda_i, \lambda_{i+1})$ takes at most $O(N^3)$ steps for NISQ-era QCs (per Appendix B) and $\|\pi_s\|$ is no greater than $4D$ which is upper-bounded by N^2 (per [35]), $O\left(c(w(\lambda_i, \lambda_{i+1}))\right) = O(N^2 + N^3) = O(N^3)$.
7	$\rightarrow O(LN^3)$	Final time complexity.

Table 36: Worst-case time complexity derivation for true objective function.

Step #	Step	Description
1	$f_{obj,u} = O(f_q) + O(f_{s,u})$	$f_{obj,u}$ is composed of two separate operations: f_q and $f_{s,u}$.
2	$\rightarrow O(f_{1q}) + O(f_{2q}) + O(f_{s,u})$	Expanding f_q.
3	$\rightarrow O\left(\prod_{\lambda_i \in s}\prod_{q_j \in T} s_1(\lambda_i, q_j)\right) + O\left(\prod_{\lambda_i \in s}\prod_{q_j \in T}\prod_{q_k \in T} s_2(\lambda_i, q_j, q_k)\right) + O(f_{s,u})$	Expanding f_{1q} and f_{2q}.
4	$\rightarrow O(LN) + O(LN^2) + O(f_{s,u})$	Given that $\|T\| = N$, $\|s\| = L$, and both s_1 and s_2 are $O(1)$ operations, $O(f_{1q}) = O(LN)$ and $O(f_{2q}) = O(LN^2)$.
5	$\rightarrow O(LN) + O(LN^2) + O(\varpi_{max}^{\varrho_l})$	Expanding $f_{s,u}$. Here, $\varrho_l = \sum_{i=1}^{L-1} \eta_l(D_i)$.
6	$\rightarrow O(LN) + O(LN^2) + O((L-1) \cdot M)$ $\rightarrow O(LN) + O(LN^2) + O(LM)$ $\rightarrow O(LN) + O(LN^2) + O(LN)$	Since ϱ_l takes at most $L-1$ steps, D_i takes at most M steps (see equation TBD), and both η_l and d_1 are $O(1)$ operations, $O(\varpi_{max}^{\varrho_l}) = O((L-1) \cdot M)$. Further, M is upper-bounded by N.
7	$\rightarrow O(LN^2)$	Final time complexity.

Table 37: Worst-case time complexity derivation for surrogate objective function.

Bibliography

[1] A. Herman, "Winning the Race in Quantum Computing," *American Affairs Journal,* vol. 2, no. 2, pp. 1-23, 2018.

[2] "Air Force Institute of Technology," 2020. [Online]. Available: https://www.afit.edu/. [Accessed 29 December 2020].

[3] Y. Zhang and Q. Yuan, "A multiple bits erro correction method based on cyclic redundancy check codes," in *International Conference on Signal Processing Proceedings, ICSP*, 2008.

[4] S. S. Tannu and M. K. Qureshi, "Not All Qubits Are Created Equal," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems - ASPLOS '19*, New York, NY, USA, 2019.

[5] D. H. Wolpert and W. G. Macready, "No Free Lunch Theorems for Optimization," *IEEE Transactions on Evolutionary Computation,* vol. 1, no. 1, pp. 67-82, 1997.

[6] E.-G. Talbi, Metaheuristics: From Design to Implementation, Hoboken, NJ: John Wiley & Sons, 2009.

[7] M. A. Nielsen and I. L. Chuang, Quantum Computation and Quantum Information, New York: Cambridge University Press, 2010.

[8] J. Schiller, Quantum Computers, BookSurge Publishing, 2009.

[9] A. Adedoyin, J. Ambrosiano, P. Anisimov, D. Bartschi, W. Casper, G. Chennupati, C. Coffrin, H. Djidjev and D. Gunter, "Quantum Algorithm Implementations for Beginners," arXiv, Los Alamos, NM, 2020.

[10] "Qiskit API Documentation," 2020. [Online]. Available: https://quantum-computing.ibm.com/.

[11] A. Zulehner, A. Paler and R. Wille, "Efficient Mapping of Quantum Circuits to the IBM QX Architectures," in *2018 Design, Automation & Test in Europe Conference & Exhibition*, 2018.

[12] P. Murali, J. M. Baker, A. J. 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*, Providence, RI, USA, 2019.

[13] M. Siraichi, F. Dos Santos, S. Collange and F. Pereira, "Qubit Allocation," *CGO 2018 - Proceedings of the 2018 International Symposium on Code Generation and Optimization,* Vols. 2018-February, pp. 113-125, 2018.

[14] X. Zhang, H. Xiang, T. Xiang, L. Fu and J. Sang, "An efficient quantum circuits optimizing scheme compared with QISKit," in *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, 2018.

[15] I. Chuang, "Lecture 19: How to Build Your Own Quantum Computer," Department of Mathematics, MIT, Cambridge, MA, USA, 2003.

[16] "IBM Quantum Experience," [Online]. Available: https://quantum-computing.ibm.com/. [Accessed 16 February 2021].

[17] B. Tan and J. Cong, "Optimality Study of Existing Quantum Computing Layout Synthesis Tools," IEEE Transactions on Computers, 2020.

[18] G. Li, Y. Ding and Y. Xie, "Tackling the Qubit Mapping Problem for NISQ-Era Quantum Devices," arXiv, Santa Barbara, CA, 2019.

[19] B. K. Kamaka, "Quantum Transpiler Optimization: On the Development, Implementation, and Use of a Quantum Research Testbed," 2020.

[20] S. Boyd and L. Vandenberghe, Convex Optimization, New York: Cambridge University Press, 2004.

[21] J. C. Bean, "Genetic Algorithms and Random Keys for Sequencing and Optimization," *ORSA Journal on Computing,* vol. 6, no. 2, pp. 154-160, 1994.

[22] N. Mladenović and J. M. Pérez, "Variable Neighborhood Search: Methods and Applications," *Annals of Operations Research,* vol. 175, no. 1, pp. 367-407, 2010.

[23] J. Lill and R. Smith, "MOES: Pareto-Based Multiple Objective Optimization Combining Evolution Strategies with Data Envelopment Analysis," 2020.

[24] J. F. Goncalves and M. G. Resende, "Biased Random-key Genetic Algorithms for Combinatorial Optimization," *Journal of Heuristics,* pp. 487-525, 2011.

[25] T. Bäck, Evolutionary Algorithms in Theory and Practice, New York, NY: Oxford University Press, 1996.

[26] K. M. Malan and A. P. Engelbrecht, "A Survey of Techniques for Characterising Fitness Landscapes and Some Possible Ways Forward," *Information Sciences,* vol. 241, pp. 148-163, 2013.

[27] R. Olson, "Sewall Wright and the Importance of Population Genetics," 2018. [Online]. Available: http://scihi.org/sewall-wright-population-genetics/. [Accessed 15 December 2020].

[28] E. D. Weinberger, "Correlated and uncorrelated fitness landscapes and how to tell the difference," *Bilogical Cybernetics,* vol. 63, pp. 325-336, 1990.

[29] S. Verel, P. Collard and M. Clergue, "Where are bottlenecks in NK fitness landscapes?," in *The 2003 Congress on Evolutionary Computation*, 2003.

[30] G. Lu, J. Li and X. Yao, "Fitness-Probability Cloud and a Measure of Problem Hardness for Evolutionary Algorithms," in *Evolutionary Computation in Combinatorial Optimization*, 2011.

[31] H. Rose, W. Ebeling and T. Asselmeyer, "The Density of States - a Measure of Difficulty of Optimisation Problems," in *PPSN IV: Proceedings of teh 4th International Conference on Parallel Problem Solving from Nature*, London, UK, 1996.

[32] F. Wickelmaier, "An Introduction to MDS," Aalborg University, Denmark, 2003.

[33] A. Ghodsi, "Metric Multidimensional Scaling (MDS)," Waterloo, Ontario, 2006.

[34] K. Yamanaka, T. Horiyama, M. Keil, D. Kirkpatrick, Y. Otachi, T. Saitoh, R. Uehara and Y. Uno, "Swapping Colored Tokens on Graphs," in *Theoretical Computer Science*, 2018.

[35] T. Miltzow, L. Narins, Y. Okamoto, G. Rote, A. Thomas and T. Uno, "Approximation and Hardness for Token Swapping," in *Algorithms-ESA 2016, Proc. 24th Annual European Symposium on Algorithms, Aarhus, 2016, Leibniz International Proceedings in Informatics*, 2016.

[36] M. Lewin, "All About XOR," June 2012. [Online]. Available: https://accu.org/journals/overload/20/109/lewin_1915/.

[37] A. E. Brownlee, J. R. Woodward and J. Swan, "Metaheuristic Design Pattern: Surrogate Fitness Functions," *GECCO 2015 - Companion Publication of the 2015 Genetic and Evolutionary Computation Conference,* no. October, pp. 1261-1264, 2015.

[38] K. Conrad, "Generating Sets," 2016.

[39] V. K. Ky, C. D'Ambriosio, Y. Hamadi and L. Liberti, "Surrogate-based methods for black-box optimization," International Transactions in Operational Research, Palaiseau, France, 2016.

[40] H.-G. Beyer and H.-P. Schwefel, "Evolution strategies - A comprehensive introduction," *Natural Computing,* vol. 1, pp. 3-52, 2002.

[41] S. Niu, A. Suau, G. Staffelbach and A. Todri-Sanial, "A Hardware-Aware Heuristic for the Qubit Mapping Problem in the NISQ Era," *IEEE Transactions on Quantum Engineering,* vol. 1, pp. 1-14, 2020.

[42] A. Zulehner and R. Willie, "Compiling SU(4) Quantum Circuits to IBM QX Architectures," in *Proceedings of the 24th Asia and South Pacific Design Automation Conference on - ASPDAC '19*, New York, 2019.

[43] P. W. Shor, "Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer," AT&T Research, 1996.

[44] E. Wilson, S. Singh and F. Mueller, "Just-in-time Quantum Circuit Transpilation Reduces Noise," arXiv, 2020.

[45] "U.S. Air Force Research Laboratory - DoD Supercomputing Resource Center," AFRL, 18 November 2020. [Online]. Available: https://www.afrl.hpc.mil/hardware/. [Accessed 02 February 2021].

[46] "Navy DSRC: High Performance Computing Systems," 9 December 2020. [Online]. Available: https://www.navydsrc.hpc.mil/hardware/index.html. [Accessed 3 February 2021].

[47] D. E. Knuth, The Art of Computer Programming, 3 ed., vol. 1, Boston, MA: Addison Wesley Longman Publishing Co., 1997, pp. 145-146.

[48] H. Prasetyo, G. Fauza, Y. Amer and S. H. Lee, "Survey on Applications of Biased-Random Key Genetic Algorithms for Solving Optimization Problems," in *2015 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM)*, Singapore, 2015.

[49] L. V. Snyder and M. S. Daskin, "A Random-key Genetic Algorithm for the Generalized Traveling Salesman Problem," *European Journal of Operational Research,* vol. 174, no. 1, pp. 38-53, 2006.

[50] R. Pierce, "Standard Deviation and Variance," 2017. [Online]. Available: https://www.mathsisfun.com/data/standard-deviation.html.

[51] C. Zhang, Y. Chen, Y. Jin, W. Ahn, Y. Zhang and E. Z. Zhang, "A Depth-Aware Swap Insertion Scheme for the Qubit Mapping Problem," arXiv, Ithaca, NY, 2020.

[52] B. Aruoba and J. Fernández-Villaverde, "A Comparison of Programming Languages in Economics," National Bureau of Economic Research, Cambridge, MA, 2014.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From – To)*
22-03-2021	Master's Thesis	Sept 2019 – Mar 2021

TITLE AND SUBTITLE

Solving the Quantum Layout Problem for NISQ-Era Quantum Computers via Metaheuristic Algorithms

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Curran Jr, Brian D., Second Lieutenant, USAF

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(S)
Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/ENG)
2950 Hobson Way, Building 640
WPAFB OH 45433-8865

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT-ENG-MS-21-M-024

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
AFRL Quantum Science & Technology
Ms. Laura Wessing
525 Brooks Road
Rome, NY 13441
COMM 315-330-2937

10. SPONSOR/MONITOR'S ACRONYM(S)

AFRL/RITQ

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
DISTRUBTION STATEMENT A. APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES
This material is declared a work of the U.S. Government and is not subject to copyright protection in the United States.

14. ABSTRACT
In the noisy intermediate-scale quantum (NISQ)-era, quantum computers (QC) are highly prone to noise-related errors and suffer from limited connectivity between their physical qubits. Circuit transformations must be made to abstract circuits to address the noise and hardware constraints of NISQ-era devices. Such transformations introduce additional gates to the original circuit, thereby reducing the circuit's overall fidelity. To address the aforementioned constraints of NISQ-era QCs, dynamic remapping procedures permute logical qubits about physical qubits of the device to increase the fidelity of operations and make operations hardware-compliant. The quantum layout problem (QLP) is the problem of mapping logical qubits of the circuit to physical qubits of the target QC in a way that maximizes circuit fidelity and satisfies all device connectivity constraints. This research effort seeks to use metaheuristic algorithms to find high-quality solutions to the QLP. In this work, the QLP is mathematically modeled, integrated into various optimization algorithm domains, and resultant algorithms evaluated for efficiency and effectiveness. Moreover, fitness landscape analysis is performed based on the devised representation, objective functions, and search operators.

15. SUBJECT TERMS
Quantum Computing, Transpiler Optimization, Metaheuristics, Qiskit

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Laurence D. Merkle, AFIT/ENG
U	U	U	UU	190	19b. TELEPHONE NUMBER *(Include area code)* (312) 785-3636 x4526 Laurence.Merkle.afit.edu

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. Z39-18