ABSTRACT

Title of Thesis: Quantum algorithms and the power of forgetting

Amin Shiraz Gilani
Master of Science in Computer Science, 2022

Thesis Directed by: Andrew Childs, Matthew Coudron
Department of Computer Science

The so-called Welded Tree Problem provides an example of a black-box problem that can be solved exponentially faster by a quantum walk than by any classical algorithm [1]. Given the name of a special ENTRANCE vertex, a quantum walk can find another distinguished EXIT vertex using polynomially many queries, though without finding any particular path from ENTRANCE to EXIT. It has been an open problem for twenty years whether there is an efficient quantum algorithm for finding such a path, or if the path-finding problem is hard even for quantum computers. We show that a natural class of efficient quantum algorithms provably cannot find a path from ENTRANCE to EXIT in the Welded Tree Problem. Specifically, we consider algorithms that, within each branch of their superposition, always store a set of vertex labels that form a connected subgraph including the ENTRANCE, and that only provide these vertex labels as inputs to the oracle. While this does not rule out the possibility of a quantum algorithm that efficiently finds a path, it is hard to imagine how an algorithm could benefit by deviating from this behavior. Our no-go result suggests that, to outperform classical computation, quantum algorithms must necessarily forget the path they take to reach a solution.

Quantum algorithms and the power of forgetting

by

Amin Shiraz Gilani

Thesis submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Master of Science in Computer Science
2022

Advisory Committee:
Dr. Andrew Childs, Chair/Advisor
Dr. Matthew Coudron, Advisor
Dr. William Gasarch

Foreword

"For certain, you have to be lost to find a place that can't be found. Elseways, everyone would know where it was." — Captain Hector Barbossa.

Dedication

To my parents! This would not have been possible without you.

Acknowledgments

I am utterly grateful to Andrew Childs and Matthew Coudron for their remarkable supervision. Throughout the course of this thesis, Andrew and Matt were always up for helpful discussions and allowed me independence whenever I felt confident. I also thank them for the giving me the opportunity to work on a challenging problem and trusting my abilities. Working with them has been very enlightening, and I hope and wish to continue doing that!

My fellows at QuICS have enhanced my graduate experience and deserve a special mention. In particular, I thank Daochen for many scientific discussions and for being an incredible inspiration. I am also grateful to Shubham, Yingkang and Connor for many engaging conversations.

I am truly indebted to my mother Nazira, my father Shiraz and my brother Salim for their unconditional support. They have stood by me in all hardships and have been a source of constant guidance. I express my tremendous gratitude to my friends for always being there, especially[1] Ambreen, Anabia, Anas, Areeba, Baqar, Barira, Bushra, Laiba, Lalarukh, Ridwa, Sumbul, Talha and Ukasha. I am also grateful to my housemates, particularly Mansur, for numerous interesting conversations and helping me through various challenges.

Finally, I would like to acknowledge financial support from the United States Educational Foundation in Pakistan and the Institute of International Education via a Fulbright scholarship.

[1] in alphabetical order

Table of Contents

List of Figures

Chapter 1: Introduction

Quantum algorithms use interference of many branches of a superposition to solve problems faster than is possible classically. Shor's factoring algorithm [2] achieves a superpolynomial speedup over the best known classical algorithms by efficiently finding the period of a modular exponentiation function, and several other quantum algorithms provide a speedup by similarly detecting periodic structures [3, 4, 5]. While a few other examples of dramatic quantum speedup are known—notably including the simulation of quantum dynamics [6]—our understanding of the capabilities of quantum algorithms remains limited. To gain more insight into the possible applications of quantum computers, we would like to better understand the kinds of problems they are able to solve efficiently and what features of problems they are able to exploit.

Another example of exponential quantum speedup is based on quantum analogs of random walks. Specifically, quantum walks provide an exponential speedup for the so-called Welded Tree Problem [1]. The symmetries of this problem, and the structure of the quantum algorithm for solving it, seem fundamentally different from all preceding exponential quantum speedups. In particular, the Welded Tree Problem provably requires polynomial "quantum depth" to solve efficiently [7], whereas all previously known exponential quantum speedups only require logarithmic quantum depth, including Shor's factoring algorithm [8]. (The only other known computational problem exhibiting an exponential quantum speedup, yet requiring polynomial quantum depth, was recently constructed in [9].)

The Welded Tree Problem is defined on a "welded tree graph" that is formed by joining

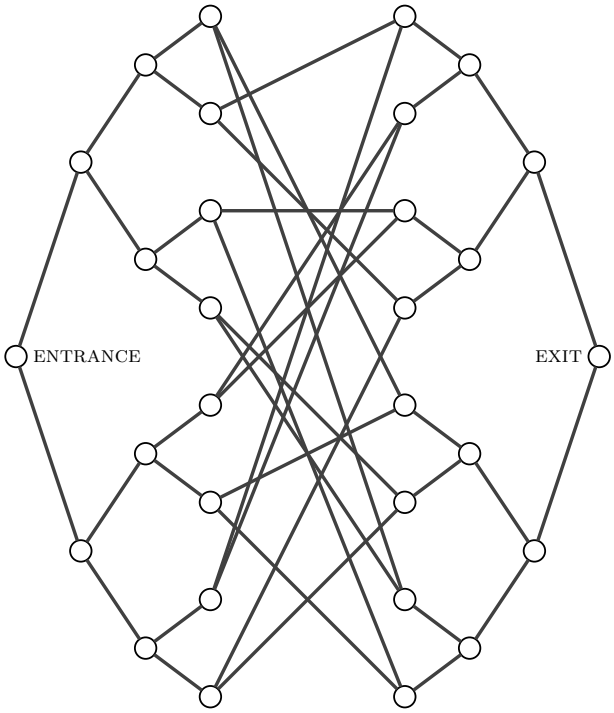


Figure 1.1: Example of a welded tree graph with $n = 3$.

the leaves of two binary trees with a cycle that alternates between them, as shown in Figure 1.1. The root of one tree is designated as the ENTRANCE, and the root of the other tree is designated as the EXIT. The graph structure is provided through an oracle that gives adjacency-list access to the graph, where the vertices are labeled arbitrarily. Given the label of the ENTRANCE vertex and access to the oracle, the goal of the Welded Tree Problem is to return the label of the EXIT vertex. On a quantum computer, this black box allows one to perform a quantum walk, whereby the graph is explored locally in superposition. Interference obtained by following many paths coherently causes the quantum walk to reach the EXIT in polynomial time. In contrast, no polynomial-time classical algorithm can efficiently find the EXIT—essentially because it cannot distinguish the welded tree graph from a large binary tree

While the quantum walk algorithm efficiently finds the EXIT by following exponentially many paths in superposition, it does not actually output any of those paths. Classical intuition might suggest that an efficient algorithm for finding the EXIT could be used to efficiently find a

path by simply recording every intermediate state of the EXIT-finding algorithm. However, in general, the intermediate state of a quantum algorithm cannot be recorded without destroying superposition and ruining the algorithm. In other words, the Welded Tree Problem can be viewed as a kind of multi-slit experiment that takes the classic double-slit experiment into the high-complexity regime. This raises a natural question: Is it possible for some quantum algorithm to efficiently find a path from the ENTRANCE to the EXIT? This question was already raised in the original paper on the Welded Tree Problem [1] and has remained open since, recently being highlighted in a survey of Aaronson [10].

In one attempt to solve this problem, Rosmanis introduced a model of "snake walks," which allow extended objects to move in superposition through graphs [11]. The state of a snake walk is a superposition of "snakes" of adjacent vertices, rather than a superposition of individual vertices as in a standard quantum walk. While Rosmanis did not show conclusively that snake walks cannot find a path through the welded tree graph, his analysis suggests that a snake walk algorithm is unlikely to accomplish this using only polynomially many queries to the Welded Tree Oracle. While this is only one particular approach, its failure supports the conjecture that it might not be possible to find a path efficiently. If such an impossiblity result could be shown for general quantum algorithms, it would establish that, in order to find the solution to some computational problems, a quantum algorithm must necessarily "forget" the path it takes to that solution. While forgetting information is a common feature of quantum algorithms, which often uncompute intermediate results to facilitate interference, many algorithms are able to efficiently produce a certificate for the solution once they have solved the problem.[1] In contrast, hardness of path finding in the Welded Tree Problem would show not only that trying to remember a path would cause one particular algorithm to fail, but in fact *no* algorithm can efficiently collect such

[1]For example, in Simon's algorithm [12], we learn the hidden string and can easily find collisions. In Shor's algorithm [2], the factors reveal the structure of the input number and their correctness can be easily checked.

information.

In this thesis, we take a step toward showing hardness of the welded tree path-finding problem. Specifically, we show hardness under two natural assumptions that we introduce in Section 2.2, namely that the algorithm is *genuine* and *rooted*.

First, we assume that the algorithm accesses the oracle for the input graph in a way that we call *genuine*. A genuine algorithm is essentially one that only provides meaningful vertex labels as inputs to the Welded Tree Oracle. Both the ordinary quantum walk [1] and the snake walk [11] can be implemented by genuine algorithms. It is hard to imagine that non-genuine behavior could be useful, although we leave formalizing this for future work.

We also assume that the algorithm is *rooted*. Informally, a rooted algorithm is one that always maintains a path from the ENTRANCE to every vertex appearing in its state. While remembering a path to the ENTRANCE limits how interference can occur, it does not eliminate quantum interference entirely. Indeed, if the snake walk of [11] were to find a path from ENTRANCE to EXIT, the most natural way of doing so would be in a rooted fashion. A non-rooted algorithm would effectively have to find the ENTRANCE again after detaching from it. While we cannot rule out this possibility, it seems implausible.

Our main result is that a genuine, rooted quantum algorithm cannot find a path from ENTRANCE to EXIT in the Welded Tree Graph using only polynomially many queries. To show this, we suppose that such a quantum algorithm exists and use it to construct a classical query algorithm that could solve the Welded Tree Problem using only polynomially many classical queries, with a certain error probability. To bound this error, we show (in an analysis reminiscent of the classical hardness result of [1], but enhanced to show hardness given additional edge-coloring information as discussed below) that our classical query algorithm cannot solve the Welded Tree Problem with more than exponentially small probability. This establishes our main

result.

We construct the classical query algorithm in this argument as follows. First, using exponential time and only a constant number of classical queries, the classical query algorithm employs a special procedure to process the circuit diagram of the assumed genuine, rooted quantum algorithm and sample a "transcript" (defined in Section 3.1) that describes a computational path the quantum algorithm could have taken, neglecting the possibility of encountering a cycle. The classical algorithm then makes polynomially many queries to the classical oracle, in a manner prescribed by the sampled transcript, and outputs the vertices of the welded tree graph that were reached by those queries. We prove in Chapter 3 that this efficient classical query algorithm is almost as likely to find an ENTRANCE–EXIT path as the original genuine, rooted quantum algorithm.

A subtle—yet unexpectedly significant—detail in our analysis is that we consider a version of the Welded Tree Problem in which the oracle provides a 3-coloring of the edges of the graph, instead of using a 9-coloring as in [1].[2] This alternative coloring scheme substantially reduces the complexity of the analysis in Chapters 2 and 3. This is because it allows us to determine, with high probability, whether starting at the ENTRANCE and following the edges prescribed by a polynomial-length color sequence $t \in \mathcal{C}^{\times q(n)}$, will lead to a valid vertex of the Welded Tree Graph, using only a constant number of classical queries to the Welded Tree Oracle. In particular, it suffices to check whether t departs from the ENTRANCE along one of the two valid edges (which can be determined using only three queries to the oracle). This is a key property used in our argument that the transcript state (see Definition 28) can track much of the behavior of a genuine rooted quantum algorithm while only making a small number of classical queries to the Welded Tree Oracle.

[2]Note that the quantum walk algorithm can solve the Welded Tree Problem using a 3-coloring, or even if it is not provided with a coloring at all [1].

However, our choice of the 3-coloring model comes at the cost of having the redesign the proof of classical hardness of finding the EXIT vertex in the Welded Tree Graph. The original classical hardness proof [1] crucially considers a special type of 9-coloring with the property that, starting from a valid coloring, the color of any edge can be altered arbitrarily, and only edges within distance 2 need to be re-colored to produce a valid coloring with that newly assigned edge color. This "local re-colorability" property is used at the crux of the classical hardness result, first in reducing from Game 2 to Game 3, and again implicitly in part (i) of the proof of Lemma 8 [1]. In contrast, a valid 3-coloring of the Welded Tree Graph does not have this "local re-colorability" property, and changing a single edge color might require a global change of many other edge colors to re-establish validity of the coloring. Thus we are forced to develop a modification of the classical hardness proof, given in Chapter 4 (which may also be of independent interest).

While our result does not definitively rule out the possibility of an efficient quantum algorithm for finding a path from ENTRANCE to EXIT in the Welded Tree Graph, it constrains the form that such an algorithm could take. In particular, it shows that the most natural application of a snake walk to the Welded Tree Problem, in which the snake always remains connected to the ENTRANCE, cannot solve the problem. While it is conceivable that a snake could detach from the ENTRANCE and later expand to connect the ENTRANCE and EXIT, this seems unlikely. More generally, non-genuine and non-rooted behavior do not intuitively seem useful for solving the problem. We hope that future work will be able to make aspects of this intuition rigorous.

Open questions This work leaves several natural open questions. The most immediate is to remove the assumption of a rooted, genuine algorithm to show unconditional hardness of finding a path (or, alternatively, to give an efficient path-finding algorithm by exploiting non-genuine or non-rooted behavior). We also think it should be possible to show classical hardness

of the general Welded Tree Problem when the oracle provides a 3-coloring. Finally, it would be instructive to find a way of instantiating the welded tree problem in an explicit (non-black box) fashion, giving a quantum speedup in a non-oracular setting.

Chapter 2: Welded tree problem and genuine rooted algorithms

We begin this chapter by precisely defining the Welded Tree EXIT-finding and path-finding problems. In Section 2.2, we formalize the definitions of genuine and rooted algorithms that we referred to in Chapter 1. Then, we briefly describe the notion of continuous-time quantum walk algorithms in Section 2.3, concluding with an overview of the EXIT-finding algorithm of [1] in Section 2.3.1 and the path-finding algorithm of [11] in Section 2.3.2.

2.1 Welded tree problem

We begin by describing the so-called Welded Tree Graph $\mathcal{G}_n$. Even though we briefly mentioned this graph in Chapter 1, we will precisely define it here for the ease of reading.

Definition 1 (Welded Tree)**.** *A graph $\mathcal{G}_n$ is a Welded Tree of size n if it is formed by joining the $2 \cdot 2^n$ leaves of two balanced binary trees of height n with a cycle that alternates between the two sets of leaves (as shown in Figure 1.1).*

In order to distinguish between the two binary trees used to form $\mathcal{G}_n$, we will call one the left and other the right binary trees used to form $\mathcal{G}_n$. For the rest of this thesis, we will refer to the input Welded Tree Graph of size n by $\mathcal{G}$. We will label vertices of $\mathcal{G}_n$ as follows.

Definition 2 (Vertex labels)**.** *Let* ENTRANCE *and* EXIT *denote the $2n$-bit strings that labels the roots of the left and the right binary trees respectively. We will use the $2n$-bit strings* NOEDGE *and* INVALID *to indicate the non-existence of an edge and to denote the output of an invalid query respectively. We also*

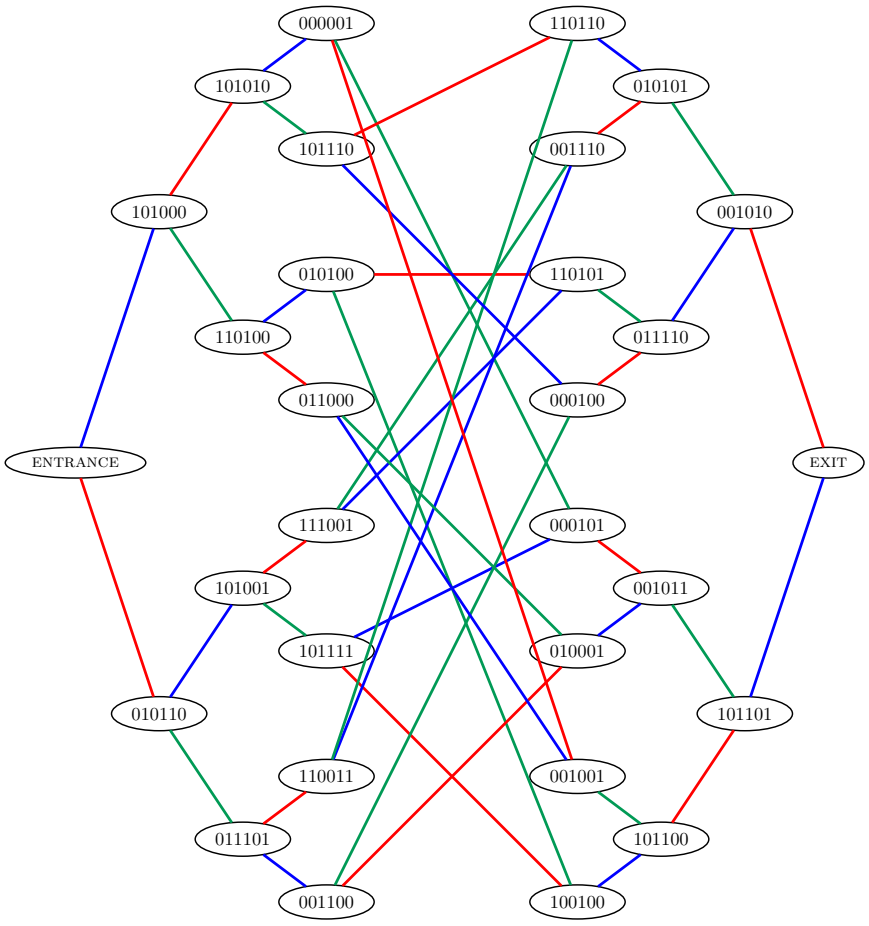


Figure 2.1: Example of a 3-colored labeled welded tree graph for $n = 3$.

define $\mathsf{SpecialVertices} := \{0^{2n}, \text{ENTRANCE}, \text{EXIT}, \text{NOEDGE}, \text{INVALID}\}$. *Each non-root vertex in* $\mathcal{G}_n$ *is labeled by an arbitrary* $2n$*-bit string not in* $\mathsf{SpecialVertices}$. *The set of vertices of* $\mathcal{G}_n$ *is denoted* $\mathcal{V}_\mathcal{G}$.

Figure 2.1 shows a valid coloring and labelling of the Welded Tree Graph in Figure 1.1. We will usually refer to the vertex labelled ENTRANCE (respectively EXIT) as ENTRANCE (respectively EXIT). Since $\mathcal{G}_n$ is bipartite and each vertex $v \in \mathcal{V}_\mathcal{G}$ has degree at most 3, it can be edge-colored using only 3 colors. Therefore, we suppose that the edges of $\mathcal{G}_n$ are colored from the set $\mathcal{C} := \{\text{red}, \text{green}, \text{blue}\}$. We define a classical oracle function $\eta_c \colon \{0,1\}^{2n} \to \{0,1\}^{2n}$ that encodes the edges of color $c \in \mathcal{C}$ in $\mathcal{G}_n$.

Definition 3 (η_c)**.** *For any* $v \in \mathcal{V}_\mathcal{G}$ *and* $c \in \mathcal{C}$*, let* $I_c(v)$ *be the indicator variable that is* 1 *if the vertex*

labeled v has an edge colored c and 0 otherwise. If $I_c(v) = 1$ for some $v \in \mathcal{V}_\mathcal{G}$ and $c \in \mathcal{C}$, let $N_c(v)$ be the label of the vertex joined to v with an edge of color c. Then

$$\eta_c(v) := \begin{cases} N_c(v) & v \in \mathcal{V}_\mathcal{G} \text{ and } I_c(v) = 1 \\ \text{NOEDGE} & v \in \mathcal{V}_\mathcal{G} \text{ and } I_c(v) = 0 \\ \text{INVALID} & v \notin \mathcal{V}_\mathcal{G}, \end{cases} \tag{2.1}$$

We will refer to the collection of η_c for $c \in \mathcal{C}$ as the oracle $O_{\mathcal{G}_n}$ *of the Welded Tree Graph $\mathcal{G}_n$. When $\mathcal{G}_n$ is clear from context, we will use O to represent the oracle $O_{\mathcal{G}_n}$ for simplicity.*

Since $\mathcal{G}_n$ is 3-colored, for any vertex label $v \in \mathcal{V}_\mathcal{G}$, $I_c(v) = 0$ only if $v \in \{\text{ENTRANCE}, \text{EXIT}\}$. For any $v \in \mathcal{V}_\mathcal{G}$ and $c \in \mathcal{C}$, if $I_c(v) = 1$, then we will call $N_c(v)$ the c-neighbor of v in $\mathcal{G}_n$.

We are ready to express the Welded Tree EXIT-finding and path-finding problems.

Definition 4 (Welded Tree EXIT-finding)**.** *Given access to an oracle O for a Welded Tree Graph $\mathcal{G}_n$ and the label* ENTRANCE*, the* Welded Tree EXIT-finding *problem requires finding the label* EXIT*. The* query complexity of a quantum (respectively classical) query algorithm $\mathcal{A}$ *solving the Welded Tree* EXIT*-finding problem is the number of quantum (respectively classical) queries made by $\mathcal{A}$ to O. The* quantum (respectively classical) query complexity of the Welded Tree EXIT-finding problem *is the minimum quantum (respectively classical) query complexity of any quantum (respectively classical) query algorithm solving the Welded Tree* EXIT*-finding problem.*

Definition 5 (Welded Tree path-finding)**.** *Given access to an oracle O for a Welded Tree Graph $\mathcal{G}_n$ and the label* ENTRANCE*, the* Welded Tree path-finding *problem requires finding a connected subgraph of $\mathcal{G}_n$ that contains the vertices with labels* ENTRANCE *and* EXIT*. The* query complexity of a quantum (respectively classical) query algorithm $\mathcal{A}$ *solving the Welded Tree path-finding problem is the number of quantum (respectively classical) queries made by $\mathcal{A}$ to O. The* quantum (respectively classical)

query complexity of the Welded Tree path-finding problem *is the minimum quantum (respectively classical) query complexity of any quantum (respectively classical) query algorithm solving the Welded Tree path-finding problem.*

In an alternate formulation of the Welded Tree path-finding problem, the goal is to find the labels of a set of vertices that form an ENTRANCE–EXIT path. However, this problem is equivalent to the one we defined in Definition 5 in terms of their query complexity: an ENTRANCE–EXIT path forms a connected subgraph of $\mathcal{G}_n$ containing the ENTRANCE and the EXIT; any known connected subgraph of $\mathcal{G}_n$ containing the ENTRANCE and the EXIT will contain an ENTRANCE–EXIT path, which can be extracted without making any queries.

Before we move on, we precisely define our notion of efficiency and the notation we will use for that.

Definition 6 (Efficiency)**.** *We say that a quantum (respectively classical) query algorithm is* efficient *for the Welded Tree* EXIT*-finding problem (respectively Welded Tree path-finding problem) if its quantum (respectively classical) query complexity is polynomial in n. In later Definitions and analysis, we will use $p(n)$ to denote any polynomial parametrized by n.*

2.2 Genuine and rooted algorithms

In this section, we precisely define the aforementioned notion of genuine, rooted quantum query algorithms. Intuitively, an algorithm is genuine if it only allows for "meaningful" operations, and it is rooted if it remains "connected to the ENTRANCE" throughout its course. We begin by describing our setup and recalling the definition of the Welded Tree Oracle.

Definition 7 (Vertex register and vertex space)**.** *A* vertex register *is a $2n$-qubit register that stores a vertex label. We consider quantum states that have exactly $p(n)$ vertex registers, and refer to the $2np(n)$-qubit space consisting of all the vertex registers as the* vertex space*.*

Any computational basis state in the vertex space stores $p(n)$ vertex labels, which might correspond to a subgraph of $\mathcal{G}$. During its execution, a quantum algorithm may want to store information that may not directly indicate the label of a vertex of $\mathcal{G}$. For this purpose, we define the notion of the workspace.

Definition 8 (Workspace and workspace register)**.** *A* workspace register *is a single-qubit register that can store arbitrary ancillary states. We allow for arbitrarily many workspace registers, and refer to the space consisting of all workspace registers as the* workspace*.*

In the following definition, we precisely describe the set of gates that any quantum query algorithm for finding an ENTRANCE–EXIT path could employ for querying, and manipulating the known, vertex labels in a meaningful way (i.e. without using the information stored in these labels).

Definition 9 (Genuine circuit)**.** *We say that a quantum circuit C is* genuine *if it is built from the following unitary gates.*

1. *Controlled-oracle query gates O_c for $c \in \mathcal{C}$ where the control qubit is in the workspace, and O_c acts on the jth and kth vertex registers for some distinct $j, k \in [p(n)] \coloneqq \{1, \ldots, p(n)\}$ as*

$$O_c \colon |v_j\rangle|v_k\rangle \mapsto |v_j\rangle|v_k \oplus \eta_c(v_j)\rangle \tag{2.2}$$

   where η_c is defined in Definition 3.

   Furthermore, in a genuine circuit, O_c can only be applied if $v_k = 0^{2n}$ or $v_k = \eta_c(v_j)$ for every v_j, v_k pair appearing in those respective registers in the superposition.

2. *Controlled-$e^{i\theta T}$ rotations for any $\theta \in [0, 2\pi)$ where the control qubit is in the workspace and the Hamiltonian T is defined, similarly to [1], to act on the jth and kth vertex registers for some distinct*

$j, k \in [p(n)]$ *as*

$$T\colon |v_j\rangle|v_k\rangle \mapsto |v_k\rangle|v_j\rangle. \tag{2.3}$$

We let $\wedge(A)$ *denote a controlled-*A *gate, so that* $\wedge(e^{i\theta T})$ *denotes the controlled-*$e^{i\theta T}$ *gate.*

3. *Equality check gates* $\mathcal{E}$*, which act on the* j*th and* k*th vertex registers for some distinct* $j, k \in [p(n)]$*, and on the* a*th workspace register for some workspace index* a*, as*

$$\mathcal{E}\colon |v_j\rangle|v_k\rangle|w_a\rangle \mapsto |v_j\rangle|v_k\rangle|w_a \oplus \delta[v_j = v_k]\rangle \tag{2.4}$$

where $\delta[P]$ *is* 1 *if* P *is true and* 0 *if* P *is false.*

4. NOEDGE *check gates* $\mathcal{N}$*, which act on the* j*th vertex register for some* $j \in [p(n)]$*, and on the* a^{th} *workspace register for some workspace index* a*, as*

$$\mathcal{N}\colon |v_j\rangle|w_a\rangle \mapsto |v_j\rangle|w_a \oplus \delta[v_j = \text{NOEDGE}]\rangle. \tag{2.5}$$

5. ZERO *check gates* $\mathcal{Z}$*, which act on the* j*th vertex register for some* $j \in [p(n)]$*, and on the* a^{th} *workspace register for some workspace index* a*, as*

$$\mathcal{Z}\colon |v_j\rangle|w_a\rangle \mapsto |v_j\rangle|w_a \oplus \delta[v_j = 0^{2n}]\rangle. \tag{2.6}$$

6. *Arbitrary two-qubit gates (or, equivalently, arbitrary unitary transformations) restricted to the workspace register.*

We would like to point out that we can efficiently check that whether the target register of an oracle query $\wedge(O_c)$ with the control vertex register storing $|v_j\rangle$ contains 0^{2n} or $\eta_c(v_j)$. Indeed, we can replace each oracle query gate O_c with the circuit shown in Figure 2.2 (where the whole

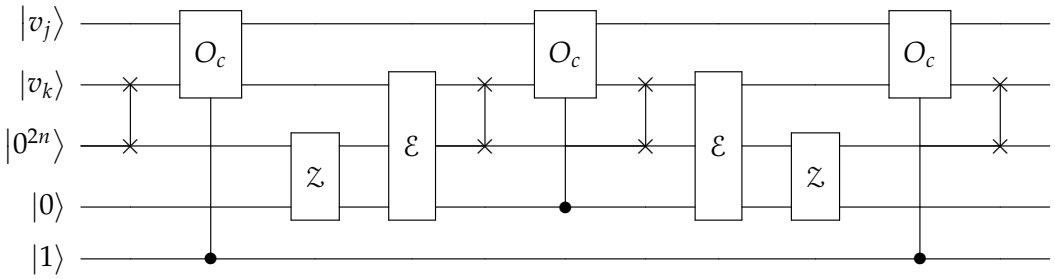


Figure 2.2: Circuit diagram for checking if $v_k = 0^{2n}$ or $v_k = \eta_c(v_j)$. The top three registers (i.e. those initialized with $|v_j\rangle$, $|v_k\rangle$ and $|0^{2n}\rangle$ respectively) are vertex registers and the bottom two registers (i.e. those initialized with $|0\rangle$ and $|1\rangle$ respectively) are workspace registers.

circuit is being controlled on the control register of O_c in the workspace) that has a constant number of gates from Definition 9. Notice that the swap gates are only used so that all the wires that a certain gate act on are adjacent. Moreover, the last workspace register allows for applying uncontrolled-O_c gates. In this circuit, the center oracle gate O_c is only applied on the registers storing $|v_j\rangle$ and $|v_k\rangle$ if the first workspace register stores a 1, which happens only when $v_k = 0^{2n}$ or $v_k = \eta_c(v_j)$ by the definitions of the ZERO check gate $\mathcal{Z}$ and the equality check gate $\mathcal{E}$. Since we are promised that the target register of any oracle gate satisfies part 1 of Definition 9, replacing each controlled-oracle gate in any given genuine circuit C with the gadget described by Figure 2.2 will not impact the output state of C and only result in a constant overhead on the size of the circuit. Therefore, without loss of generality, we can assume that the given genuine circuit checks the condition in part 1 of Definition 9 as in Figure 2.2 for each controlled-oracle gate.

We now define the notion of genuine algorithms using Definition 9. Let $O = \{O_c : c \in \mathcal{C}\}$ denote a particular randomly selected Welded Tree Oracle, and let $\mathcal{A}(O)$ denote a quantum algorithm that makes quantum queries to O.

Definition 10 (Genuine algorithm)**.** *We call a quantum query algorithm $\mathcal{A}$* genuine *if, for the given Welded Tree Oracle O, $\mathcal{A}$ acts on the vertex space and the workspace as follows.*

1. *$\mathcal{A}(O)$ begins with an initial state*

$$|\psi_{\text{initial}}\rangle = |\text{ENTRANCE}\rangle \otimes \left(|0^{2n}\rangle\right)^{\otimes(p(n)-1)} \otimes |0\rangle_{\text{workspace}}. \tag{2.7}$$

2. *Then, it applies a $p(n)$-gate genuine circuit C (as in Definition 9) on $|\psi_{\text{initial}}\rangle$ to get the state $|\psi_{\mathcal{A}}\rangle$.*

3. *Finally, it measures all the vertex registers of $|\psi_{\mathcal{A}}\rangle$ in the computational basis and outputs the corresponding vertex labels.*

Intuitively, a state in the vertex space is rooted if it corresponds to a set of labels of vertices from $\mathcal{V}_{\mathcal{G}}$ of the Welded Tree $\mathcal{G}$ described by the given oracle O (or the NOEDGE label) that form a connected subgraph containing the ENTRANCE (neglecting the NOEDGE label, if present). Formally, consider the following definition.

Definition 11 (Rooted state)**.** *We say that a computational basis state $|\psi\rangle$ in the vertex space is* rooted *if for any vertex label v stored in any of the registers of $|\psi\rangle$,*

1. *$v \in \mathcal{V}_{\mathcal{G}} \cup \{0^{2n}, \text{NOEDGE}\}$, and*

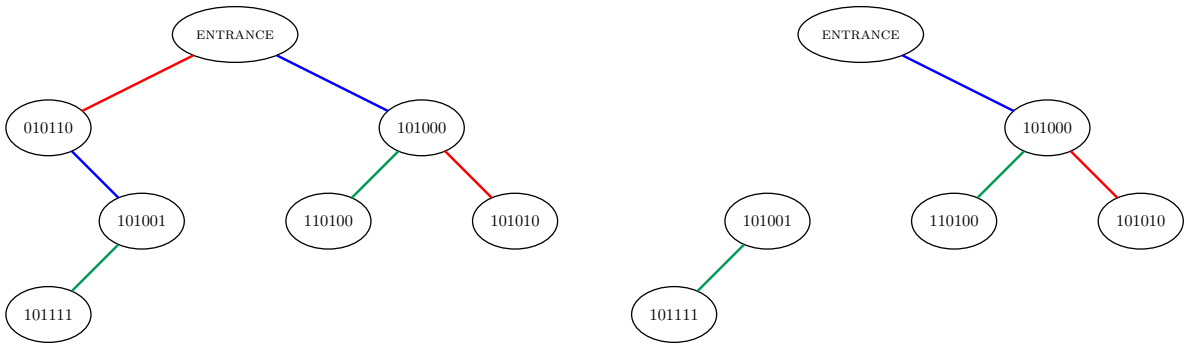
2. *if $v \notin \{0^{2n}, \text{ENTRANCE}\}$, then there exist r vertex registers storing vertex labels $v_{j_1}, \ldots, v_{j_r}$ such that $v_{j_1} = \text{ENTRANCE}$, $v_{j_r} = v$, and for each $k \in [r-1]$, $\eta_c(v_{j_k}) = v_{j_{k+1}}$ for some $c \in \mathcal{C}$.*

Figure 2.3 shows examples of rooted and non-rooted states.

Finally, we define the notion of a rooted algorithm, which is based on Definition 11.

Definition 12 (Rooted algorithm)**.** *A quantum query algorithm $\mathcal{A}$ is* rooted *if, for the given Welded Tree Oracle O, at each intermediate step of $\mathcal{A}(O)$, every computational basis state in the support of the vertex space of the quantum state maintained by $\mathcal{A}$ is rooted.*

Non-rooted behavior can be useful for exploring the Welded Tree Graph. In particular, the algorithm of [1] for finding the EXIT is not rooted, since it only maintains a single vertex (in



(a) Subgraph of Figure 2.1 corresponding to the state $|\text{ENTRANCE}\rangle$, $|010110\rangle$, $|101000\rangle$, $|101001\rangle$, $|101010\rangle$, $|101111\rangle$, $|110100\rangle$.

(b) Subgraph of Figure 2.1 corresponding to the state $|\text{ENTRANCE}\rangle$, $|000000\rangle$, $|101000\rangle$, $|101001\rangle$, $|101010\rangle$, $|101111\rangle$, $|110100\rangle$.

Figure 2.3: Examples of rooted and non-rooted states.

superposition). However, a path-finding algorithm must store information about many vertices, and the value of detaching from the ENTRANCE is unclear since it must ultimately reattach. Note that the snake walk [11] is initially rooted, the most natural way for it to find a path from ENTRANCE to EXIT is arguable to do so while remaining rooted, though the algorithm may become non-rooted if it is run for long enough.

Earlier in this section, we argued that it is possible to construct a genuine circuit that verifies that the condition given in part 1 of Definition 9 is always satisfied with a constant overhead in the size of the given circuit. Now, we will make a similar argument to demonstrate that given a quantum query algorithm, one can efficiently modify it to ensure that the state it maintains is always rooted. We do this because of technical reasons that will be evident in Sections 3.4 and 3.5.

In particular, we assert that given a genuine, rooted algorithm associated with a circuit C, one can efficiently construct a modified genuine, rooted algorithm associated with a circuit C' such that each gate G in C is replaced by a sequence of gates in C' that make sure that G is only applied if the resulting state after applying G would have been rooted with a polynomial overhead in circuit size and no impact on the resulting state. Recall from our earlier argument

that (controlled) oracle gates are the only genuine gates that can alter the contents of vertex registers. This means that we only need to replace G with this sequence of gates if G is an oracle gate. We now describe a rooted algorithm that accomplishes this task and argue how it can be implemented by a genuine circuit.

First, note that at the beginning of the circuit C, one can copy the label of the ENTRANCE (which is stored in the first register of $|\psi_{\text{initial}}\rangle$) to an ancilla vertex register that is not meant to be used by any of the gates that follow. This step can be implemented by a genuine algorithm by querying a valid neighbor of the ENTRANCE and then computing the ENTRANCE in this special ancilla vertex register before uncomputing this neighbor of the ENTRANCE. Similalrly, we can uncompute the contents of the special ancilla vertex register at the end of our algorithm. Thus, we can make sure that the ENTRANCE label is always stored in a vertex register of any computational basis state in the support of our state at any step.

Given $p(n)$ vertex registers, by a standard breadth-first search procedure starting at the ENTRANCE, one can check whether the subgraph G of $\mathcal{G}$ induced by the vertex labels stored in these registers is connected. At each step of this breadth-first search, we determine which vertex registers store the neighbors of a particular vertex v. This can be done by looping over each vertex register, checking whether it stores a neighbor of v and storing the outcome in a workspace register. Therefore, the task of checking the connectivity of G is reduced to the task of checking whether two input vertex registers stores labels of vertices that are neighbors in $\mathcal{G}$ and computing it in a workspace register.

We describe a genuine circuit for it as shown in Figure 2.4. The swap gates and the workspace register initialized to $|1\rangle$ have the same role as in Figure 2.2. Given vertex labels v_j and v_k, we first compute each of the 3 neighbors (according to η_c) of v_j in 3 different ancilla vertex registers. Then, we check whether any of these vertices are equal to v_k using equality

check gates $\mathcal{E}$, and compute the output in an ancilla workspace regsiter. By the end of this circuit, this workspace register will store 1 iff v_j and v_k are neighbors. Once we have used this workspace qubit for applying a controlled oracle gate, we uncompute the contents of this qubit by applying the circuit in Figure 2.4 backwards.

Now that we have outlined a procedure of checking rooted-ness, we describe the sequence of gates such that replacing a controlled oracle query gate O_c by this sequence ensures that each computational basis state in the support of any intermediate step of any algorithm is rooted. We first compute the output of O_c in an ancilla vertex register. Then, we determine whether O_c is meant to uncompute the contents of a particular vertex register. We check this using an equality check gate applied on the above ancilla vertex register and the target register of O_c. If O_c is not meant to uncompute, we know that applying it cannot result in a non-rooted state. In the case that it is an uncomputation, we check that the state corresponding to the collection of vertex registers not including the target register of O_c is rooted by the procedure we sketched above. We then apply O_c controlled on the output of this check and uncompute all the information we computed in ancilla vertex and workspace registers. It is easy to see that this sequence of gates never results in a non-rooted state and does not alter the output state of a genuine, rooted algorithm. Hence, without loss of generality, we can assume that any given genuine, rooted algorithm will have embedded this self-checking rooted-ness preservation mechanism.

2.3 Continuous-time quantum walk algorithms

The definition of a continuous-time quantum walk is inspired by the definition of continuous-time random walk and the Schrödinger equation. More precisely, we have the following.

Definition 13. *Let $G = (V, E)$ be a graph. Let H be a Hermitian matrix. The* continuous-time quantum walk *algorithm for matrix H is described as follows. The state $|\psi(t)\rangle$ of the evolution of this*

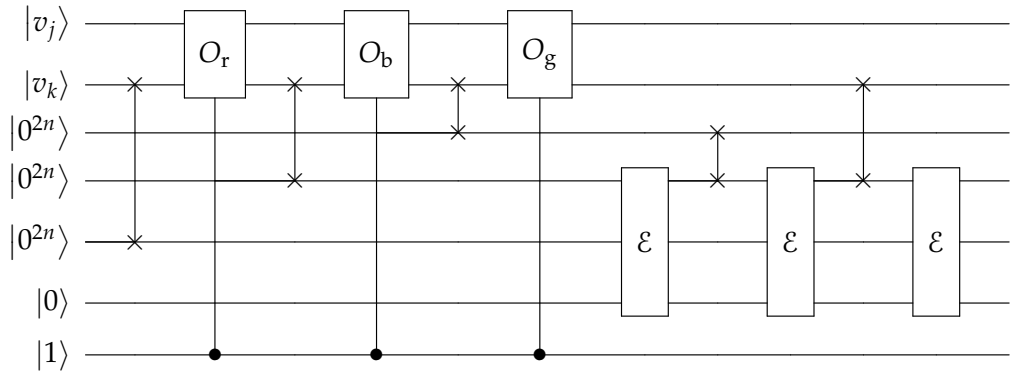


Figure 2.4: Circuit diagram for computing if $v_j = \eta_c(v_k)$ for some $c \in \mathcal{C}$. The top five registers are vertex registers and the bottom two registers are workspace registers. For compactness, we have truncated the names of the oracles O_{red}, O_{blue} and O_{green} to O_{r}, O_{b} and O_{g} respectively.

walk at time t is given by the solution of the Schrödinger equation

$$i\frac{d}{dt}|\psi(t)\rangle = H|\psi(t)\rangle \tag{2.8}$$

The matrix H describing a continuous-time quantum walk is often referred to as a *Hamiltonian* as it often describes the evolution dynamics of a physical process. The closed form solution of the state of a continuous-time quantum walk at time t is well-understood.

Lemma 14. *The closed form solution of the state $|\psi(t)\rangle$ is $e^{-itH}|\psi(0)\rangle$.*

Proof. Since H is Hermitian, it admits a spectral decomposition. That is, we can write

$$H = \sum_i \lambda_i |\lambda_i\rangle\langle\lambda_i| \tag{2.9}$$

where λ_i and $|\lambda_i\rangle$ denotes the ith eigenvalue and eigenvector respectively with the vectors in $\{\lambda_i\}_i$ form an orthonormal basis.

Since

$$He^{-itH} = \sum_i \lambda_i e^{-it\lambda}|\lambda_i\rangle\langle\lambda_i| = i\frac{d}{dt}e^{-itH}, \tag{2.10}$$

we have

$$H|\psi(t)\rangle = He^{-itH}|\psi(0)\rangle = i\frac{d}{dt}e^{-itH}|\psi(0)\rangle = i\frac{d}{dt}|\psi(t)\rangle \tag{2.11}$$

□

It can also be easily verified that e^{-itH} is a unitary operator for any Hermitian matrix H. Lemma 14 implies that it is sufficient to perform the evolution e^{-itH} in order to implement the quantum walk for the Hamiltonian H as a circuit.

Next, we allude to a continuous-time quantum walk algorithm each for the Welded Tree EXIT-finding and path-finding problems.

2.3.1 Quantum walk for Welded Tree EXIT-finding

Childs et al. [1] introduced the problem of Welded Tree EXIT-finding to demonstrate the first exponential speed-up provided by algorithms based on quantum walks. Here, we briefly describe the Hamiltonian they used and the implementation of the corresponding unitary evolution. First, consider the following definition.

Definition 15 (Adjacency matrix)**.** *Given a graph $G = (V, E)$, the adjacency matrix A of G is defined as the $|V| \times |V|$ matrix as follows. For any $j, k \in |V|$, $A_{j,k}$ is 1 if $\{j, k\} \in E$ and 0 otherwise.*

The adjacency matrix A of $\mathcal{G}_n$ acts on the jth register in the vertex space as

$$A|v_j\rangle \mapsto \delta[v \in \mathcal{V}_\mathcal{G}] \sum_{c \in \mathcal{C}: I_c(v_j)=1} |\eta_c(v_j)\rangle \tag{2.12}$$

where $I_c(v_j)$ is as defined in Definition 3.

Clearly, A is a Hermitian matrix. The quantum walk algorithm of [1] is based on the unitary evolution $e^{i\theta A}$ corresponding to A. Now, we briefly explain the implementation of $e^{i\theta A}$ using the gates described in Definition 9.

For each $c \in \mathcal{C}$, consider the Hamiltonian S_c defined to act on the jth and kth registers in the vertex space as

$$S_c|v_j\rangle|v_k\rangle \mapsto \begin{cases} |v_k\rangle|v_j\rangle & v_j \in \mathcal{V}_\mathcal{G} \wedge I_c(v_j) = 1 \\ 0 & \text{otherwise} \end{cases} \tag{2.13}$$

Observe that S_c is not a unitary. However, we will only need to apply $e^{i\theta S_c}$ for some $\theta \in [0, 2\pi)$ with $v_j \in \mathcal{V}_\mathcal{G}$. Recall that for $v \in \mathcal{V}_\mathcal{G}$ and $c \in \mathcal{C}$, $I_c(v) = 0$ iff $\eta_c(v) = \text{NOEDGE}$. The unitary evolution $e^{i\theta S_c}$ can be implemented (as shown in Figure 2.5) by computing $\eta_c(v_j)$ in an ancilla vertex register R_V by applying O_c on $|v_j\rangle$ and R_V, then computing $\delta[\eta(v_j) = \text{NOEDGE}]$ in an ancilla workspace register R_W by applying NOEDGE-check gate on R_V and R_W, then applying $e^{i\theta T}$ controlled on R_W, and finally uncomputing the contents of R_W and R_V in order. In this circuit, the swap gates and the last workspace register has the same role that they had in Figure 2.2. Notice that

$$\sum_{c\in\mathcal{C}} O_c \circ S_c \circ O_c|v_j\rangle|0^{2n}\rangle = \sum_{c\in\mathcal{C}} O_c \circ S_c|v_j\rangle|\eta_c(v_j)\rangle \tag{2.14}$$

$$= \delta[v_j \in \mathcal{V}_\mathcal{G}] \sum_{c\in\mathcal{C}:I_c(v_j)=1} O_c|\eta_c(v_j)\rangle|v_j\rangle \tag{2.15}$$

$$= \delta[v_j \in \mathcal{V}_\mathcal{G}] \sum_{c\in\mathcal{C}:I_c(v_j)=1} |\eta_c(v_j)\rangle|v_j \oplus \eta_c(\eta_c(v_j))\rangle \tag{2.16}$$

$$= \delta[v_j \in \mathcal{V}_\mathcal{G}] \sum_{c\in\mathcal{C}:I_c(v_j)=1} |\eta_c(v_j)\rangle|0^{2n}\rangle \tag{2.17}$$

$$= A|v_j\rangle|0^{2n}\rangle \tag{2.18}$$

where the penultimate step follow since $\eta_c(\eta_c(v_j)) = v_j$ as $v_j \in \mathcal{V}_\mathcal{G}$ and $I_c(v_j) = 0$.

This means that A (extended to act on one ancilla register along with one vertex register) can be written as a linear combination of the Hamiltonians $O_c \circ S_c \circ O_c$ for $c \in \mathcal{C}$. That is, we can

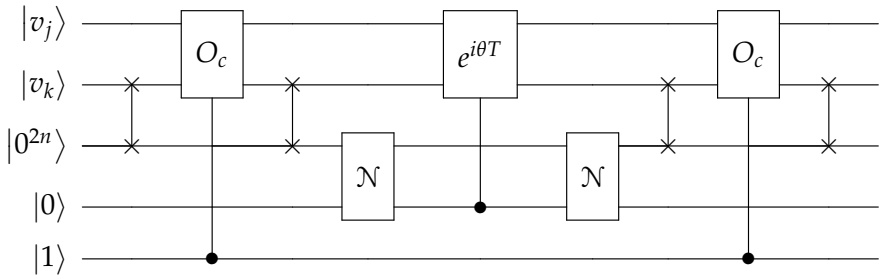


Figure 2.5: Circuit diagram for $e^{i\theta S_c}$ using genuine gates.

write the evolution operator $e^{i\theta A}$ as $e^{\sum_{c\in\mathcal{C}} O_c \circ S_c \circ O_c}$. Using known methods, one can approximate this operator with error polynomial in θ and the largest norm of the commutator of $O_c \circ S_c \circ O_c$ and $O_{c'} \circ S_c \circ O_{c'}$ for $c, c' \in \mathcal{C}$.

Now, we provide some intuition as to why the quantum walk algorithm for the Hamiltonian A works. Before that, we look at an elementary yet useful definition.

Definition 16 (Column and level)**.** *Let T be a binary tree of height n. We say that a vertex of T is in* column *j if its distance from the root of T is j, and an edge in T is at* level *j if it connects a vertex in column $j-1$ to a vertex in column j. We extend this notion to the Welded Tree Graphs. Precisely, a vertex of a Welded Tree Graph $\mathcal{G}$ is in* column *j if its distance from the* ENTRANCE *is j, and an edge in $\mathcal{G}$ is at* level *j if it connects a vertex in column $j-1$ to a vertex in column j.*

In particular, the EXIT is in column $2n+1$ and the edges connecting the leaves of the left and the right binary trees are at level $n+1$. Note that the number of vertices in column $j \in [2n+1] \cup \{0\}$ are

$$\nu_j = \begin{cases} 2^j & j \leq n \\ 2^{2n-j+1} & \text{otherwise} \end{cases} \tag{2.19}$$

while the number of edges at level $j \in [2n+1]$ are

$$\mu_j = \begin{cases} 2^j & j \leq n+1 \\ 2^{2n-j+2} & \text{otherwise} \end{cases} \tag{2.20}$$

Let us define $|\mathsf{col}(j)\rangle$ to be a uniform superposition over all vertices in column j of $\mathcal{G}_n$. Precisely,

$$|\mathsf{col}(j)\rangle = \frac{1}{\sqrt{\nu_j}} \sum_{v:v \text{ in column } j} |v\rangle \tag{2.21}$$

Note that for any $j, k \in [2n-1] \cup \{0\}$ with $|j-k| > 1$, it must be that $\langle \mathsf{col}(j)|A|\mathsf{col}(k)\rangle = 0$ since no vertex in column j is joined by an edge to any vertex in column k. For any $j \in [2n] \cup \{0\}$, we have

$$\langle \mathsf{col}(j)|A|\mathsf{col}(j+1)\rangle = \frac{\mu_{j+1}}{\sqrt{\nu_j}\sqrt{\nu_{j+1}}} = \begin{cases} 2 & j = n \\ \sqrt{2} & \text{otherwise} \end{cases} \tag{2.22}$$

Similarly, for any $j \in [2n-1]$, we have

$$\langle \mathsf{col}(j)|A|\mathsf{col}(j-1)\rangle = \frac{\mu_j}{\sqrt{\nu_j}\sqrt{\nu_{j-1}}} = \begin{cases} 2 & j = n+1 \\ \sqrt{2} & \text{otherwise} \end{cases} \tag{2.23}$$

It follows that the subspace span $|\mathsf{col}(j) : j \in [2n-1] \cup \{0\}\rangle$ is invariant under A. Moreover, we can view A as the adjacency matrix of a weighted path graph whose vertices correspond to states in $|\mathsf{col}(j)\rangle$. As the weights are almost uniform, one can expect the quantum walk using A as its Hamiltonian beginning at $|\mathsf{col}(0)\rangle = |\text{ENTRANCE}\rangle$ to have inverse polynomial (possibly inverse linear) amplitude on the state $|\mathsf{col}(j)\rangle$ for each $j \in [2n-1]$ after polynomially (possible

linearly) many steps.

2.3.2 Quantum snake walk for Welded Tree path-finding

In this subsection, we mention some key ideas about the quantum snake walk algorithm due to Rosmanis [11]. We begin by describing an expansion of the Welded Tree Graph that the Rosmanis' quantum snake walk algorithm will run on.

Definition 17 (Expanded Welded Tree Graph)**.** *Let $\mathcal{G}$ be any Welded Tree Graph and r be an integer greater than 0. The* Expanded Welded Tree Graph $\mathcal{G}^{(r)}$ *associated with $\mathcal{G}$ and r is constructed by making 2^r copies of $\mathcal{G}$ and connecting them such that the* ENTRANCE *vertex of each of these copies is a leaf of a binary tree T_{ENTRANCE} of height r and the* EXIT *vertex of each of these copies is a leaf of a separate binary tree T_{EXIT} of height r. Let $\mathcal{V}_{\mathcal{G}^{(r)}}$ denote the set of vertices of $\mathcal{G}^{(r)}$.*

Observe that the root of T_{ENTRANCE} is distance $2n + 2r + 1$ away from the root of T_{EXIT}. We now describe the formalism needed to define the Welded Tree Snake Graph, whose vertices correspond to paths in the Welded Tree Graph.

Definition 18 (Snakes)**.** *Let $\mathcal{G}$ be any Welded Tree Graph, and r and m be integers greater than 0. A* snake *of length m in $\mathcal{G}^{(r)}$ is a tuple $(v_1, \ldots, v_{m+1})$ of $m+1$ vertices such that $v_i \in \mathcal{V}_{\mathcal{G}^{(r)}}$ for each $i \in [m+1]$ and there is an edge between v_i and v_{i+1} in $\mathcal{G}^{(r)}$ for each $i \in [m]$. The vertices v_{m+1} and v_1 are called the* head *and the* tail *of the snake $(v_i, \ldots, v_{m+1})$. Let $\mathcal{V}_{\mathbb{S}_{m,r}(\mathcal{G})}$ denote the set of all snakes of length m in $\mathcal{G}^{(r)}$.*

Equivalently, a snake of length m in $\mathcal{G}^{(r)}$ can be described as the collection of an initial vertex and a sequence of m edges such that the first edge in this sequence is incident on the initial vertex and each consecutive edge share a vertex that they are incident on. Note that it is possible to have $v_i = v_j$ for $i \neq j$.

Definition 19 (Movement of snakes)**.** *Let $\mathcal{G}$ be any Welded Tree Graph, and r and m be integers greater than 0. Let $(v_1, \ldots, v_{m+1})$ be any length m snake in $\mathcal{G}^{(r)}$. Let v be any neighbor of v_{m+1} in $\mathcal{G}^{(r)}$ and let u be any neighbor of v_1 in $\mathcal{G}^{(r)}$. Then, we say that the snake $(v_1, \ldots, v_{m+1})$ moves* forward *to the snake $(v_2, \ldots, v_{m+1}, v)$. Similarly, we say that the snake $(v_1, \ldots, v_{m+1})$ moves* backward *to the snake $(u, v_1, \ldots, v_m)$.*

Observe that a snake $\vec{v} = (v_1, \ldots, v_{m+1})$ moves forward to the snake $\vec{u} = (u_1, \ldots, u_{m+1})$ iff $\vec{u}$ moves backward to the snake $\vec{v}$. The following definition describes a graph whose vertices are labeled by paths in the Expanded Welded Tree Graph of Definition 17.

Definition 20 (Welded Tree Snake Graph)**.** *Let $\mathcal{G}$ be any Welded Tree Graph, and r and m be integers greater than 0. The* Welded Tree Snake Graph $\mathcal{S}_{m,r}(\mathcal{G})$ *is the graph with vertex set $\mathcal{V}_{\mathcal{S}_{m,r}(\mathcal{G})}$ whose edges defined as follows. For any snakes $\vec{v} = (v_1, \ldots, v_{m+1})$ and $\vec{u} = (u_1, \ldots, u_{m+1})$ in $\mathcal{G}^{(r)}$, there is an edge of length 1 between $\vec{v}$ and $\vec{u}$ if either $\vec{v}$ moves forward to $\vec{u}$ or $\vec{u}$ moves forward to $\vec{v}$ but not both, and there is an edge of length 2 between $\vec{v}$ and $\vec{u}$ if $\vec{v}$ moves forward to $\vec{u}$ and $\vec{u}$ moves forward to $\vec{v}$. We say that the vertex corresponding to the snake $\vec{v} = (v_1, \ldots, v_{m+1}) \in \mathcal{V}_{\mathcal{S}_{m,r}(\mathcal{G})}$ is marked if there are $j, k \in [m+1]$ such that $v_j =$ ENTRANCE and $v_k =$ EXIT.*

In order for $\mathcal{S}_{m,r}(\mathcal{G})$ to have any marked vertex, m must be at least $2n + 1$. Any marked vertex of $\mathcal{S}_{m,r}(\mathcal{G})$ will correspond to a snake that contains an ENTRANCE–EXIT path so the goal now is to find a marked vertex of $\mathcal{S}_{m,r}(\mathcal{G})$. For this purpose, it is sufficient to find a vertex of $\mathcal{S}_{m,r}(\mathcal{G})$, which corresponds to a snake that contains a vertex each from T_{ENTRANCE} and T_{EXIT}. Continuous-time quantum snake walk, which is defined next, provides an algorithm for finding such a vertex.

Definition 21 (Quantum snake walk)**.** *Let $\mathcal{G}$ be any Welded Tree Graph, and r and m be integers greater than 0. The continuous-time quantum snake walk on $\mathcal{G}^{(r)}$ with Hamiltonian H and initial state $|\psi(0)\rangle$ is*

defined as the quantum walk on the graph $\mathcal{S}_{m,r}(\mathcal{G})$ with Hamiltonian H and initial state $|\psi(0)\rangle$.

Rosmanis [11] describes a quantum snake walk algorithm for finding a marked vertex in $\mathcal{S}_{m,r}(\mathcal{G})$. We provide an overview of this algorithm as follows.

Algorithm 1: Rosmanis quantum snake walk algorithm for the Welded Tree path-finding problem

---

1 Pick any m, r polynomial in n with $m \geq 2n + 1$.

2 Prepare an initial state $|\psi(0)\rangle$. [1]

3 Run the quantum snake walk on $\mathcal{G}^{(r)}$ with the Hamiltonian being the adjacency matrix of $\mathcal{G}^{(r)}$ and initial state $|\psi(0)\rangle$ for some polynomial (in n) number of steps.

4 Measure the final state of this walk to obtain a snake s of length m.

5 **if** *s contains a vertex each from T_{ENTRANCE} and T_{EXIT}* **then**

6 | **return** s

7 **else**

8 | Go back to Line 2

Note that this algorithm only terminates once a snake has been found that contains a vertex labeled ENTRANCE and a vertex labeled EXIT. Such a snake would contain an ENTRANCE–EXIT path by Definitions 17 and 18 in the Expanded Welded Tree graph. Therefore, Line 1 is correct. However, the query complexity of this algorithm is not clear. In fact, no bounds on this complexity were shown in [11] and it is still unknown.

[1] Expressing this state would require more machinery, which we believe is a digression for our purpose of communicating the intuition of this algorithm so we omit it.

Chapter 3: Classical simulation of genuine, rooted algorithms

We begin this chapter by describing the notion of transcript states in Section 3.1 that emulate the state of the given genuine, rooted algorithm. Then, we define a mapping that sends states in the address space to states in the vertex space in Section 3.2. In Section 3.3, we state our classical simulation algorithm of genuine, rooted algorithms. Then, we define states that would help us in our analysis of this algorithm in Section 3.4. We show in Section 3.5 that the 'good' part of the quantum state of a genuine, rooted algorithm is related, via the mapping L defined in Definition 32, to the 'good' part of the state of our simulation algorithm at each intermediate step. Finally, we establish in Section 3.6 that any genuine, rooted algorithm cannot find an ENTRANCE–EXIT path (or a cycle) with more than exponentially small probability using the result of Chapter 4.

3.1 Transcript states

With each genuine quantum query algorithm $\mathcal{A}(O)$ that makes $p(n)$ oracle queries to the oracle O of the input welded tree $\mathcal{G}$, we associate a quantum state $|\phi_{\mathcal{A}}\rangle$, which we call the *transcript state* of $\mathcal{A}(O)$. As we will see in Definition 28, instead of storing the label of a vertex v, any register of any computational basis state of the transcript state $|\phi_{\mathcal{A}}\rangle$ will store a path from the ENTRANCE to v. We will formally refer to this path as the 'address' of v, which is defined next.

Definition 22 (Vertex addresses)**.** *We say that a tuple t of colors from $\mathcal{C}$ is an* address *of a vertex*

v of $\mathcal{G}$ if v is reached by starting at the ENTRANCE *and following the edge colors listed in t. For completeness, we assign special names* ZEROADDRESS, NOEDGEADDRESS, *and* INVALIDADDRESS *to denote the addresses of vertex labels 0^{2n},* NOEDGE, *and* INVALID, *respectively. We denote the empty tuple by the special name* EMPTYADDRESS. *Let* SpecialAddresses $:=$ {ZEROADDRESS, EMPTYADDRESS, NOEDGEADDRESS, INVALIDADDRESS}. *We define*

$$\text{ADDRESSES} := \textit{SpecialAddresses} \cup \bigcup_{i \in [p(n)]} \mathcal{C}^{\times i} \tag{3.1}$$

where $\mathcal{C}^{\times i}$ denotes the set of all i-tuples of colors from $\mathcal{C}$.

Note that a given vertex can have many different associated addresses. Indeed, two addresses that differ by an even-length palindrome of colors are associated to the same vertex. Even greater multiplicity of addresses can occur because of the cycles in $\mathcal{G}$. We define the notion of the *address tree* to deal with the former issue, and we delay consideration of the latter issue. To define the address tree, we need to know the color c_* that does not appear at the ENTRANCE.

Definition 23 (The bad color at the entrance)**.** *Let $c_* \in \mathcal{C}$ be the unique color such that there is no edge of color c_* incident to the* ENTRANCE *in $\mathcal{G}$.*

Definition 24 (Address tree)**.** *The* address tree *$\mathcal{T}$ (see Figure 3.1) is a binary tree of depth $p(n)$ with 3 additional vertices.*[1] *Its vertices and edges are labeled by addresses and colors, respectively, as follows. The 3 additional vertices are labeled by each address in* SpecialAddresses $\setminus$ {EMPTYADDRESS}. *The root of $\mathcal{T}$ is labeled by* EMPTYADDRESS. *It is joined to the vertex labeled* NOEDGEADDRESS *by a directed edge of color c_*, and to 2 other vertices, each by an undirected edge of a distinct color from $\mathcal{C} \setminus \{c_*\}$. For each color $c \in \mathcal{C}$, the vertices labeled* ZEROADDRESS *and* NOEDGEADDRESS *have a directed edge colored c to the vertex labeled* INVALIDADDRESS. *The vertex labeled* INVALIDADDRESS *also has 3 self-loop edges,*

[1] The address tree is not technically a tree, but we still refer to it as such because it contains no non-trivial cycles.

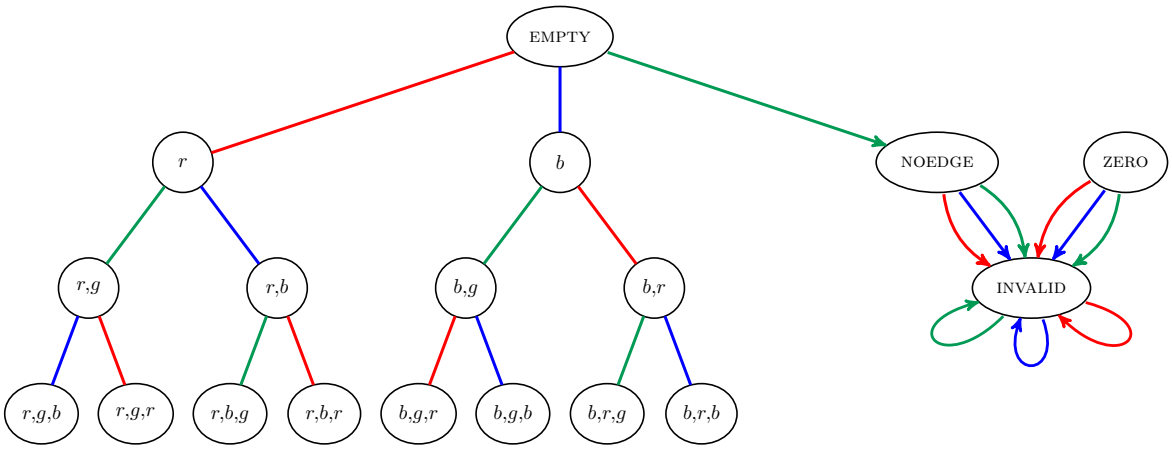


Figure 3.1: Address tree $\mathcal{T}$ of depth 3 corresponding to the graph in Figure 2.1. For the sake of brevity, we have removed the suffix ADDRESS for all the addresses in SpecialAddresses and the tuple brackets for all the addresses not in SpecialAddresses. Notice that, for each vertex, there is an edge (either directed or undirected) of each color outgoing from each vertex in $\mathcal{T}$.

each of a distinct color from $\mathcal{C}$*. Every other vertex in* t *is joined to 3 other vertices, each by an undirected edge of a distinct color from* $\mathcal{C}$*. Every vertex* t *of* $\mathcal{T}$ *whose label is not in* SpecialAddresses *is labeled by the sequence of colors that specifies the (shortest) path from* EMPTYADDRESS *to* t *in* $\mathcal{T}$*. For any vertex* t *of* $\mathcal{T}$*, let* $\lambda_c(t)$ *be the vertex that is joined to* t *by an edge of color* c *in* $\mathcal{T}$*.*

The following simple observations about the address tree $\mathcal{T}$ may be instructive.

- Since the 3-coloring of $\mathcal{T}$ is a valid coloring, no vertex label of $\mathcal{T}$ will ever contain an even-length palindrome.

- Beginning at the vertex labeled EMPTYADDRESS and traversing any sequence of colors in $\mathcal{T}$ will lead us to some vertex of $\mathcal{T}$. Therefore, in the definition of the transcript state (Definition 28), and hence in the algorithm analyzed in Sections 3.3 to 3.6, the addresses that we consider are valid labels of vertices in $\mathcal{T}$, by construction.

- The color c_* can be computed with 2 queries to the oracle O. Therefore, the entire address tree can be computed with only 2 queries to O.

Intuitively, the transcript state $|\phi_{\mathcal{A}}\rangle$ is the state that results from running the algorithm $\mathcal{A}$

on the address tree $\mathcal{T}$. If $\mathcal{A}$ does not explore cycles in the welded tree $\mathcal{G}$ to a significant extent, then $|\phi_{\mathcal{A}}\rangle$ should be a good approximation of the state $|\psi_{\mathcal{A}}\rangle$ produced by running $\mathcal{A}$ on $\mathcal{G}$, as in Definition 10. We show, in Sections 3.3 to 3.6, that this is indeed the case for any genuine, rooted quantum algorithm $\mathcal{A}$.

Now we define a mapping B that turns addresses into strings, and another mapping B^{inv} that turns strings into addresses, such that B^{inv} is the inverse of B on the range of B. We will later note that the registers that we consider can never contain any string that is no in the range of the B mapping. Therefore, it is sufficient to define B^{inv} over the range of B. Nevertheless, we will define B^{inv} over $\{0,1\}^{2p(n)}$ for the sake of completeness.

Definition 25 (B mapping)**.** *Let $\mathcal{V}_{\mathcal{T}}$ denote the set of labels of vertices of the address tree $\mathcal{T}$. Let S be a subset of $\{0,1\}^{2p(n)}$ of size $|\mathcal{V}_{\mathcal{T}}|$ containing $0^{2p(n)}$. Let* EMPTYSTRING, NOEDGESTRING, *and* INVALIDSTRING *be any fixed distinct strings in $S \setminus \{0^{2p(n)}\}$. Then $B\colon \mathcal{V}_{\mathcal{T}} \to S$ is a bijection mapping* ZEROADDRESS *to* $0^{2p(n)}$, EMPTYADDRESS *to* EMPTYSTRING, NOEDGEADDRESS *to* NOEDGESTRING, *and* INVALIDADDRESS *to* INVALIDSTRING. *We define the function $B^{\mathsf{inv}}\colon \{0,1\}^{2p(n)} \to \mathcal{V}_{\mathcal{T}}$ as*

$$B^{\mathsf{inv}}(s) := \begin{cases} B^{-1}(s) & s \in S \\ \text{INVALIDADDRESS} & \textit{otherwise.} \end{cases} \tag{3.2}$$

We will now define analogs of the spaces we defined in Definitions 7 and 8 that our transcript state (Definition 28) will lie in and our classical simulation algorithm (Algorithm 3) will act on.

Definition 26 (Address register and address space)**.** *An* address register *is a $2p(n)$-qubit register storing bit strings that are the image, under the map B, of the address of some vertex label in the address tree $\mathcal{T}$. We consider quantum states that have exactly $p(n)$ address registers, and refer to the $2p(n)^2$-qubit*

space of all the address registers as the address space.

Definition 27 (Address workspace and address workspace register)**.** *An* address workspace register *is a single-qubit register that stores arbitrary ancillary states. We allow for arbitrarily many address workspace registers, and refer to the space consisting of all address workspace registers as the* address workspace.

Notice the similarity between the definitions of workspace and address workspace. Indeed, we will later notice that the projection of $|\psi_{\mathcal{A}}\rangle$ on the workspace is the same as the projection of $|\psi_{\mathcal{A}}\rangle$ on the address workspace in the subspace not containing the EXIT or a cycle. We are now ready to state the definition of the transcript state associated with the quantum state $|\psi\rangle_{\mathcal{A}}$.

Definition 28 (Transcript state)**.** *Consider a $p(n)$-query genuine, rooted quantum algorithm $\mathcal{A}$. Given a circuit C that acts on the vertex space and the workspace, let $\tilde{C}$ be the quantum circuit that acts on the address space and the address workspace, obtained by the following procedure.*[2]

1. *Determine c_* using two queries to the oracle O.*

2. *Replace each vertex register with an address register and each workspace register with an address workspace register. Replace the initial state used the genuine algorithm (recall Definition 10) with the new initial state*

$$|\phi_{\text{initial}}\rangle := |\text{EMPTYSTRING}\rangle \otimes \left(\left|0^{2p(n)}\right\rangle\right)^{\otimes(p(n)-1)} \otimes |0\rangle_{\text{addressworkspace}}. \tag{3.3}$$

In parts 3 and 8 below, we describe gates that act on the address space analogously to how the gates in Definition 9 act on the vertex space. For any vertex $v \in \mathcal{V}_{\mathcal{G}}$, we write $s_v \in \{0,1\}^{2p(n)}$ to denote the contents of the address register corresponding to the vertex register storing v. The transcript

[2]Notice that the time complexity of this procedure is linear in the size of the circuit C.

state is produced by the unitary operation that results by replacing each vertex-space gate in the quantum algorithm $\mathcal{A}$ with the corresponding address-space gate defined below.

3. *Replace any controlled-oracle gate in C (controlled on workspace register a and acting on vertex registers j and k) with controlled-$\tilde{O}_c$ (controlled on address workspace register a and acting on address registers j and k), where*

$$\tilde{O}_c\colon |s_j\rangle|s_k\rangle \mapsto |s_j\rangle|s_k \oplus B(\lambda_c(B^{\mathsf{inv}}(s_j)))\rangle. \tag{3.4}$$

4. *Replace any controlled-$e^{i\theta T}$ gate in C (controlled on workspace register a and acting on vertex registers j and k) with a controlled-$e^{i\theta \tilde{T}}$ gate (controlled on address workspace register a and acting on address registers j and k), where*

$$\tilde{T}\colon |s_j\rangle|s_k\rangle \mapsto |s_k\rangle|s_j\rangle. \tag{3.5}$$

5. *Replace any equality check gate $\mathcal{E}$ in C (controlled on vertex registers j and k and acting on workspace register a) with $\tilde{\mathcal{E}}$ (controlled on address registers j and k and acting on address workspace register a), where*

$$\tilde{\mathcal{E}}\colon |s_j\rangle|s_k\rangle|w_a\rangle \mapsto |s_j\rangle|s_k\rangle|w_a \oplus \delta[s_j = s_k]\rangle. \tag{3.6}$$

6. *Replace any* NOEDGE*-check gate $\mathcal{N}$ in C (controlled on vertex register j and acting on workspace register a) with $\tilde{\mathcal{N}}$ (controlled on address register j and acting on address workspace register a), where*

$$\tilde{\mathcal{N}}\colon |s_j\rangle|w_a\rangle \mapsto |s_j\rangle|w_a \oplus \delta[s_j = \textsc{noedgestring}]\rangle. \tag{3.7}$$

7. *Replace any* ZERO*-check gate $\mathcal{Z}$ in C (controlled on vertex register j and acting on workspace register*

a) with $\tilde{\mathcal{Z}}$ (controlled on address register j and acting on address workspace register a), where

$$\tilde{\mathcal{Z}}\colon |s_j\rangle|w_a\rangle \mapsto |s_j\rangle\Big|w_a \oplus \delta[s_j = 0^{2p(n)}]\Big\rangle. \tag{3.8}$$

8. *Leave gates acting on the workspace unchanged.*

The transcript state $|\phi_{\mathcal{A}}\rangle$ *is obtained by applying the circuit* $\tilde{C}$ *to the string* EMPTYSTRING $=$ $B(\text{EMPTYADDRESS})$*, together with* $p(n)-1$ *ancilla address registers storing* $0^{2n} = B(\text{ZEROADDRESS})$*. In other words,*

$$|\phi_{\mathcal{A}}\rangle \coloneqq \tilde{C}|\phi_{\text{initial}}\rangle. \tag{3.9}$$

Notice that whereas C updates the vertex registers by making many oracle queries to O, the circuit $\tilde{C}$ only makes two queries to O.

Recall from section 2.2 that we assumed that the given genuine circuit C will have built-in gadgets described by Figure 2.2 that verifies the condition asserted in part 1 of Definition 9. Since we construct $\tilde{C}$ from C by a gate-by-gate process, we apply an oracle gate $\tilde{O}_c$ if the target register in the workspace of the gates $\tilde{\mathcal{Z}}$ and $\tilde{\mathcal{E}}$ just before the gate $\wedge(\tilde{O}_c)$ (as in Figure 2.2) is in the state $|1\rangle$. Therefore, by the definition of the $\tilde{\mathcal{Z}}$ and $\tilde{\mathcal{E}}$ gates, the oracle gate $\wedge(\tilde{O}_c)$ acting on address registers storing $|s_j\rangle$ and $|s_k\rangle$ is only applied when $s_k = 0^{2p(n)}$ or $s_k = B(\lambda_c(B^{\mathsf{inv}}(s_j))$ (and the control qubit of $\wedge(\tilde{O}_c)$ in the address workspace in the state $|1\rangle$). Notice that all the other gates in Definition 28 either does not alter the address registers at all or shuffles their positions without changing the address strings stored. It follows that no gate from Definition 28 will introduce address labels that are not in the range of the B mapping defined in Definition 25. Since $|\phi_{\text{initial}}\rangle$ does not store any address labels not in the range of B, applying any circuit C composed of the

gates from Definition 28 to $|\phi_{\text{initial}}\rangle$ will not result in a state that stores any address labels not in the range of B.

3.2 Mapping addresses to vertices

In this section, we define an efficiently computable function L' that takes an address t as input and outputs the corresponding vertex label v, and observe some relationships of addresses and the vertices they map to under L'. For $t \in \{\text{ZEROADDRESS}, \text{INVALIDADDRESS}\}$, this function simply outputs the corresponding vertex label. Otherwise, the image of t under L' is obtained by performing a sequence of oracle calls to determine the vertices reached by following edges of the colors specified by t, and outputting the vertex label reached at the end of that sequence. More precisely, $L'(t)$ is computed as follows.

Algorithm 2: Classical Query Algorithm for Computing $L'(t)$

Input: An address t

Output: A vertex label v

1 **if** $t = \text{ZEROADDRESS}$ **then**
2 **return** 0^{2n}
3 **if** $t = \text{INVALIDADDRESS}$ **then**
4 **return** INVALID
5 $v \leftarrow \text{ENTRANCE}$;
6 **for** $i = 1 \ldots |t|$ **do**
7 $v \leftarrow \eta_{t[i]}(v)$;
8 **return** v;

Here $|t|$ denotes the length of the address t (the number of colors in its color sequence), and $t[i]$ denotes the ith color. We now consider immediate implications of the definition of the mapping L' in Line 2, beginning from the following Lemma, which states that any address of

ENTRANCE that is not the EMPTYADDRESS encodes a cycle in $\mathcal{G}$.

Lemma 29. *Let $t \neq$ EMPTYADDRESS be such that $L'(t) =$ ENTRANCE. Then traversing the edge colors listed in t beginning from the* ENTRANCE *yields a cycle in $\mathcal{G}$.*

Proof. Since $L'(t) =$ ENTRANCE and $t \neq$ EMPTYADDRESS, we know that $t \notin$ SpecialAddresses. Therefore, t can be written as a sequence of edge colors. As we noted earlier, this sequence of colors does not contain any even-length palindrome. This means beginning at the ENTRANCE in $\mathcal{G}$ and following the edge colors listed in t does not involve any backtracking. Moreover, as $L'(t) =$ ENTRANCE, traversing this sequence of colors results in reaching the ENTRANCE. Therefore, this sequence in $\mathcal{G}$ starting and ending at ENTRANCE forms a cycle. □

We can generalize Lemma 29 to show that any two distinct addresses of any vertex label in $\mathcal{V}_\mathcal{G} \cup$ SpecialVertices together encodes the address of the EXIT or a cycle in $\mathcal{G}$ as follows.

Lemma 30. *Let t and t' be addresses with $t \neq t'$ and $L'(t) = L'(t')$. If $L'(t) = L'(t') \notin$ SpecialVertices, then beginning from the* ENTRANCE *in $\mathcal{G}$ and following the edge colors listed in t in order and then following the edge colors listed in t' in reverse order forms a path that contains a non-trivial cycle in $\mathcal{G}$. Otherwise, there is a $\tau \in \{t, t'\}$ such that beginning from the* ENTRANCE *in $\mathcal{G}$ and following the edge colors listed in τ will result in either reaching the* EXIT *or forming a path that contains a cycle in $\mathcal{G}$.*

Proof. Let $v = L'(t) = L'(t')$. We consider six cases:

1. $v = 0^{2n}$. Note that, by Line 2, $\tau =$ ZEROADDRESS is the only address τ such that $L'(\tau) = 0^{2n}$. Therefore, $t = t' =$ ZEROADDRESS so this case is not possible.

2. $v =$ ENTRANCE. Since $t \neq t'$, either $t \neq$ EMPTYADDRESS or $t' \neq$ EMPTYADDRESS. In either case, the result follows from Lemma 29.

3. $v = \text{EXIT}$. Then for both $\tau \in \{t, t'\}$, beginning from the ENTRANCE in $\mathcal{G}$ and following the edge colors listed in τ will result in reaching the EXIT.

4. $v = \text{NOEDGE}$. Since $t \neq t'$, either $t \neq \text{NOEDGEADDRESS}$ or $t' \neq \text{NOEDGEADDRESS}$. Without loss of generality, let $t \neq \text{NOEDGEADDRESS}$. This means that $t \notin \mathsf{SpecialAddresses}$ so it can be written as a non-empty sequence $(c_1, \ldots, c_{|t|})$ of colors. Let τ be the address specified by the color sequence $(c_1, \ldots, c_{|t|-1})$. By Line 2, $\eta_{c_{|t|}}(L'(\tau)) = L'(t) = v$. Since $v = \text{NOEDGE}$, we have $L'(\tau) \in \{\text{ENTRANCE}, \text{EXIT}\}$ by Definition 3. If $L'(\tau) = \text{EXIT}$, then following the edge colors in τ, and hence in t, results in reaching the EXIT, so we are done. It remains to consider $L'(\tau) = \text{ENTRANCE}$. It must be that $\tau \neq \text{EMPTYSTRING}$; otherwise, t would be NOEDGEADDRESS. Thus, by Lemma 29, following the edge colors in τ, and hence in t, beginning from the ENTRANCE forms cycle in $\mathcal{G}$.

5. $v = \text{INVALID}$. Since $t \neq t'$, either $t \neq \text{INVALIDADDRESS}$ or $t' \neq \text{INVALIDADDRESS}$. Without loss of generality, let $t \neq \text{INVALIDADDRESS}$. Then $t \notin \mathsf{SpecialAddresses}$, so it can be written as a non-empty sequence $(c_1, \ldots, c_{|t|})$ of colors. Let τ be the address specified by the color sequence $(c_1, \ldots, c_{|t|-1})$. By Line 2, $\eta_{c_{|t|}}(L'(\tau)) = L'(t) = v$. Since $v = \text{NOEDGE}$, we have $L'(\tau) \in \{0^{2n}, \text{NOEDGE}, \text{INVALID}\}$ by Definition 3. By Line 2, $L'(\tau) = 0^{2n}$ only when $\tau = \text{ZEROADDRESS}$. But we know that $\tau \neq \text{ZEROADDRESS}$, so we cannot have $L'(\tau) = 0^{2n}$. If $L'(\tau) = \text{NOEDGE}$, then the desired result follows from part 4 of Lemma 30. If $L'(\tau) = \text{INVALID}$, then we let $t = \tau$ and can apply the same argument recursively. We conclude that following the edge colors in τ, and hence in t, beginning from the ENTRANCE forms a non-trivial cycle in $\mathcal{G}$.

6. $v \notin \mathsf{SpecialVertices}$. This means that $t, t' \notin \mathsf{SpecialAddresses}$, so we can write $t = (c_1, \ldots, c_{|t|})$ and $t' = (c'_1, \ldots, c'_{|t'|})$ as non-empty sequences of colors. Let τ be the address

specified by the color sequence $(c_1, \ldots, c_{|t|}, c'_{|t'|}, \ldots c'_1)$ formed by concatenating the sequence t with the sequence t' in reverse order. Since v is a valid vertex of $\mathcal{G}$, following this sequence in $\mathcal{G}$ beginning with ENTRANCE will result in reaching the ENTRANCE (via v). That is, $L'(\tau) = \text{ENTRANCE}$. Moreover, as t and t' are vertex labels in the address tree $\mathcal{T}$, following the sequence given by τ in $\mathcal{T}$ beginning with the vertex labeled EMPTYADDRESS will not result in EMPTYADDRESS; otherwise, $t = t'$. Our desired result follows by Lemma 29.

Since these cases cover all possible v, the result follows. □

The following Lemma is critical for the proof of many results that lead up to Lemma 53. Informally, it states that in the case that any address t does not encode an ENTRANCE–EXIT path or a cycle in $\mathcal{G}$, then the c-neighbor of the vertex corresponding to t in $\mathcal{G}$ is the same as the vertex corresponding to the c-neighbor of t in $\mathcal{T}$.

Lemma 31. *Let v be any vertex label, let t be an address of v, and let $c \in \mathcal{C}$. Furthermore, if $t \notin$ SpecialAddresses, suppose that following the edge colors given by t starting at the* ENTRANCE *does not result in reaching the* EXIT *or finding a cycle in $\mathcal{G}$. Then $L'(\lambda_c(t)) = \eta_c(v)$.*

Proof. As t is the address of v, we know that $L'(t) = v$ by Definition 22 and Line 2. It remains to show that $L'(\lambda_c(t)) = \eta_c(L'(t))$.

First, suppose $t \in$ SpecialAddresses. Then we have four cases:

1. $t =$ ZEROADDRESS. Then

$$L'(\lambda_c(\text{ZEROADDRESS})) = L'(\text{INVALIDADDRESS}) = \text{INVALID} \tag{3.10}$$

$$= \eta_c(0^{2n}) = \eta_c L'(\text{ZEROADDRESS}) \tag{3.11}$$

where the first step follows from Definition 24, the second and fourth steps follow from

Line 2, and the third step follows from Definition 3.

2. $t = \text{EMPTYSTRING}$. Then

$$L'(\lambda_c(\text{EMPTYSTRING})) = L'((c)) = \eta_c(\text{ENTRANCE}) = \eta_c L'(\text{EMPTYSTRING}) \tag{3.12}$$

where the first step follows from Definition 24, the second and fourth steps follow from Line 2, and the third step follows from Definition 3.

3. $t = \text{NOEDGEADDRESS}$. This case follows by an argument analogous to part 1.

4. $t = \text{INVALIDADDRESS}$. This case also follows by an argument analogous to part 1.

Therefore, assuming $t \in \mathsf{SpecialAddresses}$, $L'(\lambda_c(t)) = \text{INVALID} = \eta_c(L'(t))$.

Now, suppose $t \notin \mathsf{SpecialAddresses}$. This means we can write t as a sequence $(c_1, \ldots, c_{|t|})$ of colors.

We claim that v is a label of a degree-3 vertex of $\mathcal{G}$. For the sake of contradiction, assume that $v \in \mathsf{SpecialVertices}$. In that case, Lemma 30 implies that beginning from the ENTRANCE in $\mathcal{G}$ and following the edge colors listed in t will result in either reaching the EXIT or forming a path that contains a cycle in $\mathcal{G}$, which directly contradicts the hypotheses of the lemma.

Since v is a degree-3 vertex of $\mathcal{G}$, we have $\eta_c(\eta_c(v)) = v$ by Definition 9. Therefore,

$$L'(\lambda_c(t)) = L'(\lambda_c((c_1, \ldots, c_{|t|}))) \tag{3.13}$$

$$= \begin{cases} L'((c_1, \ldots, c_{|t|-1})) & c = c_{|t|} \\ L'((c_1, \ldots, c_{|t|}, c)) & \text{otherwise} \end{cases} \tag{3.14}$$

$$= \begin{cases} (\eta_{c_{|t|-1}} \circ \cdots \circ \eta_{c_1})(v) & c = c_{|t|} \\ (\eta_c \circ \eta_{c_{|t|}} \circ \cdots \circ \eta_{c_1})(v) & \text{otherwise} \end{cases} \tag{3.15}$$

$$= (\eta_c \circ \eta_{c_{|t|}} \circ \cdots \circ \eta_{c_1})(v) \tag{3.16}$$

$$= \eta_c(L'((c_1, \ldots, c_{|t|})) \tag{3.17}$$

$$= \eta_c(L'(t)) \tag{3.18}$$

where the second step follows form Definition 24, the third and fifth from Line 2, and the fourth from an observation made above. □

3.3 The classical algorithm

Our goal for this section is to describe our classical algorithm (Algorithm 3) for simulating genuine quantum algorithms. First, we state some definitions that would help us in this goal, beginning with a map that is based on the function L' we discussed in Section 3.2.

Definition 32. *For any $m \in [p(n)]$, the mapping $L\colon \left(\{0,1\}^{2p(n)}\right)^{\otimes m} \to \left(\{0,1\}^{2n}\right)^{\otimes m}$ sends m address strings to m vertex labels as $L := (L'B^{\mathsf{inv}})^{\otimes m}$.*

When considering the map L applied to a quantum state $|\chi\rangle$ on both the workspace and the address space, we use the shorthand $L|\chi\rangle$ to denote the state $(I_{\text{workspace}} \otimes L_{\text{vertex}})|\chi\rangle$, with the map acting as the identity on the workspace register and as L on the vertex register. For the

description of Algorithm 3 and its analysis, we need to consider individual gates and consecutive sequences of gates of the genuine circuit C defined in Definition 10. For this purpose, we consider the following definition.

Definition 33. *For any $i \in [p(n)]$, let C_i denote the ith gate of the circuit C in Definition 10. For any $i, j \in [p(n)] \cup \{0\}$ with $i < j$, Let $C_{i,j}$ be the subsequence of gates from the circuit C starting with the $(i+1)$st gate and ending with the jth gate. That is, $C_{i,j} = C_j \cdots C_{i+1}$. Similarly, using the circuit $\tilde{C}$ constructed in Definition 28, we define the $\tilde{C}_i$ and $\tilde{C}_{i,j}$ for each $i, j \in [p(n)] \cup \{0\}$ with $i < j$.*

Note that $C_{i,i} = I$ and $C_{i-1,i} = C_i$ (respectively $\tilde{C}_{i,i} = I$ and $\tilde{C}_{i-1,i} = \tilde{C}_i$) for all $i \in [p(n)]$. We use the gates $C_{i,j}$ from Definition 33 in the next definition.

Definition 34. *For each $i \in [p(n)] \cup \{0\}$, let $\left|\phi_{\mathcal{A}}^{(i)}\right\rangle$ be the transcript state for the quantum algorithm $\mathcal{A}$ consisting of the first i gates of $\tilde{C}$. Equivalently,*

$$\left|\phi_{\mathcal{A}}^{(i)}\right\rangle := \tilde{C}_{0,i}|\phi_{\text{initial}}\rangle. \tag{3.19}$$

Similarly, let $\left|\psi_{\mathcal{A}}^{(i)}\right\rangle$ denote the state of the quantum algorithm $\mathcal{A}$ consisting of the first i gates of C. Equivalently,

$$\left|\psi_{\mathcal{A}}^{(i)}\right\rangle := C_{0,i}|\psi_{\text{initial}}\rangle. \tag{3.20}$$

The state $\left|\phi_{\mathcal{A}}^{(p(n))}\right\rangle = |\phi_{\mathcal{A}}\rangle$ is the transcript state corresponding to the quantum state $\left|\psi_{\mathcal{A}}^{(p(n))}\right\rangle = |\psi_{\mathcal{A}}\rangle$ in Definition 10. Consider the following classical query algorithm for finding a path from the ENTRANCE to the EXIT.

Algorithm 3: Classical query algorithm $\mathcal{C}(\mathcal{A}(O))$

for $i \in [p(n)]$ **do**

  Given the circuit diagram $C_{0,i}$, compute the transcript state $\left|\phi_{\mathcal{A}}^{(i)}\right\rangle$ as per Definition 28.

  Sample a computational basis state $\left|\phi^{(i)}\right\rangle$ in the address space at random with probability $\left\|\left\langle\phi^{(i)}\middle|\phi_{\mathcal{A}}^{(i)}\right\rangle\right\|^2$.

  Compute the computational basis state $L\left|\phi^{(i)}\right\rangle$ in the vertex space.

  Output the labels of the vertices in $L\left|\phi^{(i)}\right\rangle$.

Note that when $\mathcal{A}$ is genuine and rooted, the output of Algorithm 3 must be a connected subgraph of $\mathcal{G}$ containing the ENTRANCE. Therefore, if the output of Algorithm 3 contains the EXIT, it must also contain an ENTRANCE-to-EXIT path. In the remainder of this chapter, we show that the output of Algorithm 3 contains the EXIT (or a cycle) with exponentially small probability.

3.4 The good, the bad, and the ugly

In this section, we define states $\left|\psi_{\text{good}}^{(i)}\right\rangle$, $\left|\psi_{\text{bad}}^{(i)}\right\rangle$, and $\left|\psi_{\text{allbad}}^{(i)}\right\rangle$, which are components of the state $\left|\psi_{\mathcal{A}}^{(i)}\right\rangle$. Intuitively, $\left|\psi_{\text{good}}^{(i)}\right\rangle$ represents the portion of the state of the algorithm after i steps that has never encountered the EXIT or a near-cycle at any point in its history, $\left|\psi_{\text{bad}}^{(i)}\right\rangle$ represents the portion of the state of the algorithm after i steps that just encountered the EXIT or a near-cycle at the ith step, and $\left|\psi_{\text{allbad}}^{(i)}\right\rangle$ combines the portions of the state of the algorithm after i steps that encountered the EXIT or a near-cycle at some point in its history. To formally define these states, we introduce the notion of good and bad states, which we define as follows.

Definition 35. *We say that a computational basis state $|\phi\rangle$ in the address space is* ϕ-bad *if the subgraph corresponding to $L|\phi\rangle$ contains the* EXIT *or is at most one edge away from containing a cycle. A computational basis state $|\phi\rangle$ in the address space is* ϕ-good *if it is not* ϕ-bad*. That is, $L|\phi\rangle$ does not contain the* EXIT *and is more than one edge away from containing a cycle.*

Similarly, a computational basis state $|\psi\rangle$ in the vertex space is ψ-bad *if the subgraph corresponding to $|\psi\rangle$ contains the* EXIT *or is at most one edge away from containing a cycle and is* ψ-good *if it is not* ψ-bad.

These good and bad states span the good and bad subspaces, respectively.

Definition 36. *We define the* ϕ-BAD *subspace as*

$$\phi\text{-BAD} := \text{span}\{|\phi\rangle : |\phi\rangle \textit{ is a } \phi\text{-bad } \textit{state}\}. \tag{3.21}$$

The ϕ-GOOD *subspace is*

$$\phi\text{-GOOD} := \text{span}\{|\phi\rangle : \forall |\phi'\rangle \in \phi\text{-BAD}, \langle\phi'|\phi\rangle = 0\} = \text{span}\{|\phi\rangle : |\phi\rangle \textit{ is a } \phi\text{-good } \textit{state}\}. \tag{3.22}$$

Let Π^{ϕ}_{bad} *and* Π^{ϕ}_{good} *denote the projectors onto* ϕ-BAD *and* ϕ-GOOD*, respectively.*

The subspaces ψ-BAD *and* ψ-GOOD*, and the projectors* Π^{ψ}_{bad} *and* Π^{ψ}_{good}*, are defined analogously.*

Notice that $\Pi^{\phi}_{\text{bad}}\Pi^{\phi}_{\text{good}} = \Pi^{\phi}_{\text{good}}\Pi^{\phi}_{\text{bad}} = 0$ and $\Pi^{\phi}_{\text{bad}} + \Pi^{\phi}_{\text{good}} = I$. Similarly, $\Pi^{\psi}_{\text{bad}}\Pi^{\psi}_{\text{good}} = \Pi^{\psi}_{\text{good}}\Pi^{\psi}_{\text{bad}} = 0$ and $\Pi^{\psi}_{\text{bad}} + \Pi^{\psi}_{\text{good}} = I$. We now define the states $\left|\psi^{(i)}_{\text{good}}\right\rangle$, $\left|\psi^{(i)}_{\text{bad}}\right\rangle$ and $\left|\psi^{(i)}_{\text{allbad}}\right\rangle$ that we mentioned in the beginning of this section.

Definition 37. *We define*

$$\left|\phi^{(i)}_{\text{good}}\right\rangle := \Pi^{\phi}_{\text{good}}\left(\left|\phi^{(i)}_{\mathcal{A}}\right\rangle - C_i\left|\phi^{(i-1)}_{\text{allbad}}\right\rangle\right), \qquad \left|\phi^{(i)}_{\text{bad}}\right\rangle := \Pi^{\phi}_{\text{bad}}\left(\left|\phi^{(i)}_{\mathcal{A}}\right\rangle - C_i\left|\phi^{(i-1)}_{\text{allbad}}\right\rangle\right) \tag{3.23}$$

where

$$\left|\phi^{(i)}_{\text{allbad}}\right\rangle := \sum_{j=1}^{i} C_{j,i}\left|\phi^{(j)}_{\text{bad}}\right\rangle. \tag{3.24}$$

Moreover, let $|\phi_{\text{good}}\rangle := \left|\phi^{(p(n))}_{\text{good}}\right\rangle$*, and* $|\phi_{\text{allbad}}\rangle := \left|\phi^{(p(n))}_{\text{allbad}}\right\rangle$*. For each* $i \in [p(n)]$*, we define*

$\left|\psi^{(i)}_{\text{good}}\right\rangle, \left|\psi^{(i)}_{\text{bad}}\right\rangle, \left|\psi^{(i)}_{\text{allbad}}\right\rangle, |\psi_{\text{good}}\rangle$ *and* $|\psi_{\text{allbad}}\rangle$ *analogously.*

We now observe some properties that can be deduced from Definitions 36 and 37 that may not be immediately obvious from these definitions. We will use many of these properties throughout the rest of our analysis.

Lemma 38. *Let* $i \in [p(n)] \cup \{0\}$. *Then*

1. $\Pi^{\phi}_{\text{bad}}\left|\phi^{(i)}_{\text{bad}}\right\rangle = \left|\phi^{(i)}_{\text{bad}}\right\rangle$ *and* $\Pi^{\phi}_{\text{good}}\left|\phi^{(i)}_{\text{good}}\right\rangle = \left|\phi^{(i)}_{\text{good}}\right\rangle$,
2. $\Pi^{\psi}_{\text{bad}}\left|\psi^{(i)}_{\text{bad}}\right\rangle = \left|\psi^{(i)}_{\text{bad}}\right\rangle$ *and* $\Pi^{\psi}_{\text{good}}\left|\psi^{(i)}_{\text{good}}\right\rangle = \left|\psi^{(i)}_{\text{good}}\right\rangle$,
3. $\left|\phi^{(i)}_{\text{good}}\right\rangle$ *has disjoint support from* $\left|\phi^{(i)}_{\text{bad}}\right\rangle$,
4. $\left|\psi^{(i)}_{\text{good}}\right\rangle$ *has disjoint support from* $\left|\psi^{(i)}_{\text{bad}}\right\rangle$,
5. $\left|\phi^{(i)}_{\text{good}}\right\rangle = \left|\phi^{(i)}_{\mathcal{A}}\right\rangle - \left|\phi^{(i)}_{\text{allbad}}\right\rangle$,
6. $\left|\psi^{(i)}_{\text{good}}\right\rangle = \left|\psi^{(i)}_{\mathcal{A}}\right\rangle - \left|\psi^{(i)}_{\text{allbad}}\right\rangle$,
7. $\tilde{C}_i\left|\phi^{(i-1)}_{\text{good}}\right\rangle = \left|\phi^{(i)}_{\text{good}}\right\rangle + \left|\phi^{(i)}_{\text{bad}}\right\rangle$,
8. $C_i\left|\psi^{(i-1)}_{\text{good}}\right\rangle = \left|\psi^{(i)}_{\text{good}}\right\rangle + \left|\psi^{(i)}_{\text{bad}}\right\rangle$,
9. $\Pi^{\phi}_{\text{bad}}\left|\phi^{(i)}_{\mathcal{A}}\right\rangle = \left|\phi^{(i)}_{\text{bad}}\right\rangle + \Pi^{\phi}_{\text{bad}}C_i\left|\phi^{(i-1)}_{\text{allbad}}\right\rangle$,
10. $\Pi^{\psi}_{\text{bad}}\left|\psi^{(i)}_{\mathcal{A}}\right\rangle = \left|\psi^{(i)}_{\text{bad}}\right\rangle + \Pi^{\psi}_{\text{bad}}C_i\left|\psi^{(i-1)}_{\text{allbad}}\right\rangle$,
11. $\Pi^{\phi}_{\text{good}}\left|\phi^{(i)}_{\mathcal{A}}\right\rangle = \left|\phi^{(i)}_{\text{good}}\right\rangle + \Pi^{\phi}_{\text{good}}C_i\left|\phi^{(i-1)}_{\text{allbad}}\right\rangle$, *and*
12. $\Pi^{\psi}_{\text{good}}\left|\psi^{(i)}_{\mathcal{A}}\right\rangle = \left|\psi^{(i)}_{\text{good}}\right\rangle + \Pi^{\psi}_{\text{good}}C_i\left|\psi^{(i-1)}_{\text{allbad}}\right\rangle$.

Proof. All the statements are trivially true for $i = 0$ by Definitions 36 and 37. We show each of them separately for all $i \in [p(n)]$ assuming that they are true for $i - 1$.

1. Note that any computational basis state in the support of $\left|\phi_{\text{bad}}^{(i)}\right\rangle$ is ϕ-bad and any computational basis state in the support of $\left|\phi_{\text{good}}^{(i)}\right\rangle$ is ϕ-good by Definition 37. The desired statement follows from Definition 36.

2. Similar to the proof of part 1.

3. Since Π_{bad}^{ϕ} and Π_{good}^{ϕ} are orthogonal projectors, this follows from part 1.

4. Similar to the proof of part 3.

5. Note that

$$\left|\phi_{\text{good}}^{(i)}\right\rangle = \Pi_{\text{good}}^{\phi}\left(\left|\phi_{\mathcal{A}}^{(i)}\right\rangle - C_i\left|\phi_{\text{allbad}}^{(i-1)}\right\rangle\right) \tag{3.25}$$

$$= (I - \Pi_{\text{bad}}^{\phi})\left(\left|\phi_{\mathcal{A}}^{(i)}\right\rangle - C_i\left|\phi_{\text{allbad}}^{(i-1)}\right\rangle\right) \tag{3.26}$$

$$= \left|\phi_{\mathcal{A}}^{(i)}\right\rangle - C_i\left|\phi_{\text{allbad}}^{(i-1)}\right\rangle - \left|\phi_{\text{bad}}^{(i)}\right\rangle \tag{3.27}$$

$$= \left|\phi_{\mathcal{A}}^{(i)}\right\rangle - \left|\phi_{\text{allbad}}^{(i)}\right\rangle \tag{3.28}$$

   where we used Definition 37 in all steps except for the second one, where we used Definition 36.

6. Similar to the proof of part 5.

7. Note that

$$\tilde{C}_i\left|\phi_{\text{good}}^{(i-1)}\right\rangle = \tilde{C}_i\left(\left|\phi_{\mathcal{A}}^{(i-1)}\right\rangle - \left|\phi_{\text{allbad}}^{(i-1)}\right\rangle\right) \tag{3.29}$$

$$= \left|\phi_{\mathcal{A}}^{(i)}\right\rangle - \tilde{C}_i\left|\phi_{\text{allbad}}^{(i-1)}\right\rangle \tag{3.30}$$

$$= \left|\phi_{\text{good}}^{(i)}\right\rangle + \left|\phi_{\text{allbad}}^{(i)}\right\rangle - \tilde{C}_i\left|\phi_{\text{allbad}}^{(i-1)}\right\rangle \tag{3.31}$$

$$= \left|\phi_{\text{good}}^{(i)}\right\rangle + \left|\phi_{\text{bad}}^{(i)}\right\rangle \tag{3.32}$$

   where we used part 5 in steps 1 and 3, and Definition 37 in step 4.

8. Similar to the proof of part 7.

9. Note that

$$\Pi^{\phi}_{\text{bad}}\left|\phi^{(i)}_{\mathcal{A}}\right\rangle = \Pi^{\phi}_{\text{bad}}\left(\left|\phi^{(i)}_{\text{good}}\right\rangle + \left|\phi^{(i)}_{\text{allbad}}\right\rangle\right) \tag{3.33}$$

$$= \Pi^{\phi}_{\text{bad}}\left|\phi^{(i)}_{\text{allbad}}\right\rangle \tag{3.34}$$

$$= \Pi^{\phi}_{\text{bad}}\left(\left|\phi^{(i)}_{\text{bad}}\right\rangle + \tilde{C}_i\left|\phi^{(i-1)}_{\text{allbad}}\right\rangle\right) \tag{3.35}$$

$$= \left|\phi^{(i)}_{\text{bad}}\right\rangle + \Pi^{\phi}_{\text{bad}}\tilde{C}_i\left|\phi^{(i-1)}_{\text{allbad}}\right\rangle \tag{3.36}$$

   where we used part 5 in step 1, part 1 in steps 2 and 4, and Definition 37 in step 3.

10. Similar to the proof of part 9.

11. Note that

$$\Pi^{\psi}_{\text{good}}\left|\phi^{(i)}_{\mathcal{A}}\right\rangle = \Pi^{\phi}_{\text{good}}\left(\left|\phi^{(i)}_{\text{good}}\right\rangle + \left|\phi^{(i)}_{\text{allbad}}\right\rangle\right) \tag{3.37}$$

$$= \left|\phi^{(i)}_{\text{good}}\right\rangle + \Pi^{\phi}_{\text{good}}\left|\phi^{(i)}_{\text{allbad}}\right\rangle \tag{3.38}$$

$$= \left|\phi^{(i)}_{\text{good}}\right\rangle + \Pi^{\phi}_{\text{good}}\left(\left|\phi^{(i)}_{\text{bad}}\right\rangle + \tilde{C}_i\left|\phi^{(i-1)}_{\text{allbad}}\right\rangle\right) \tag{3.39}$$

$$= \left|\phi^{(i)}_{\text{good}}\right\rangle + \Pi^{\phi}_{\text{good}}\tilde{C}_i\left|\phi^{(i-1)}_{\text{allbad}}\right\rangle \tag{3.40}$$

   where we used part 5 in step 1, part 1 in steps 2 and 4, and Definition 37 in step 3.

12. Similar to the proof of part 11. □

We argued at the end of Section 2.2 that, without loss of generality, any consecutive sequence of gates in the circuit corresponding to the given genuine, rooted algorithm will map a rooted state to a rooted state. Notice that $\left|\psi^{0}_{\text{good}}\right\rangle = |\psi_{\text{initial}}\rangle$ and $\left|\psi^{(0)}_{\text{bad}}\right\rangle = 0$. That is, $\left|\psi^{0}_{\text{good}}\right\rangle$ and $\left|\psi^{0}_{\text{bad}}\right\rangle$ are rooted states by eq. (2.7). Thus, since $C_i\left|\psi^{(i-1)}_{\text{good}}\right\rangle = \left|\psi^{(i)}_{\text{good}}\right\rangle + \left|\psi^{(i)}_{\text{bad}}\right\rangle$ by part 8 of

Lemma 38, any computational basis state in the support of $\left|\psi^{(i)}_{\text{good}}\right\rangle + \left|\psi^{(i)}_{\text{bad}}\right\rangle$ will be rooted for each $i \in [p(n)]$. But, by part 4 of Lemma 38, $\left|\psi^{(i)}_{\text{good}}\right\rangle$ has disjoint support from $\left|\psi^{(i)}_{\text{bad}}\right\rangle$ so any computational basis state in the support of $\left|\psi^{(i)}_{\text{good}}\right\rangle$ or in the support of $\left|\psi^{(i)}_{\text{bad}}\right\rangle$ will be rooted for each $i \in [p(n)]$. We will use this deduction in Section 3.5.

Based on the intuitive description of $\left|\psi^{(i)}_{\text{good}}\right\rangle$ and $\left|\psi^{(i)}_{\text{bad}}\right\rangle$ that we provided earlier, it is anticipated that the size of the portion of the state $\left|\psi^{(i)}_{\mathcal{A}}\right\rangle$ that never encountered the EXIT or an almost cycle at any point in its history and the size of the respective portions of the state $\left|\psi^{(i)}_{\mathcal{A}}\right\rangle$ that encountered the EXIT or a cycle at the ith or earlier steps to sum to the size of $\left|\psi^{(i)}_{\mathcal{A}}\right\rangle$. The following Lemma formalizes this intuition.

Lemma 39. *Let $i \in [p(n)] \cup \{0\}$. Then $\left\|\left|\psi^{(i)}_{\text{good}}\right\rangle\right\|^2 + \sum_{j\in[i]} \left\|\left|\psi^{(j)}_{\text{bad}}\right\rangle\right\|^2 = 1$.*

Proof. We prove this claim by induction on i. The base case is trivial as $\left\|\left|\psi^{(0)}_{\text{good}}\right\rangle\right\| = 1$ and $\left\|\left|\psi^{(0)}_{\text{bad}}\right\rangle\right\| = 0$. Now, suppose that the claim is true for some i with $i+1 \in [p(n)]$. Note that

$$\left\|\left|\psi^{(i)}_{\text{good}}\right\rangle\right\|^2 = \left\|C_i\left|\psi^{(i)}_{\text{good}}\right\rangle\right\|^2 \tag{3.41}$$

$$= \left\|\left|\psi^{(i+1)}_{\text{good}}\right\rangle + \left|\psi^{(i+1)}_{\text{bad}}\right\rangle\right\|^2 \tag{3.42}$$

$$= \left\|\left|\psi^{(i+1)}_{\text{good}}\right\rangle\right\|^2 + \left\|\left|\psi^{(i+1)}_{\text{bad}}\right\rangle\right\|^2 \tag{3.43}$$

where we use the fact that the unitary C_i preserves the norm in the first step, and parts 4 and 8 of Lemma 38 in the second and third steps, respectively.

Therefore,

$$\left\| \left| \psi_{\text{good}}^{(i+1)} \right\rangle \right\|^2 + \sum_{j \in [i+1]} \left\| \left| \psi_{\text{bad}}^{(j)} \right\rangle \right\|^2 = \left\| \left| \psi_{\text{good}}^{(i)} \right\rangle \right\|^2 - \left\| \left| \psi_{\text{bad}}^{(i+1)} \right\rangle \right\|^2 + \sum_{j \in [i+1]} \left\| \left| \psi_{\text{bad}}^{(j)} \right\rangle \right\|^2 \tag{3.44}$$

$$= \left\| \left| \psi_{\text{good}}^{(i)} \right\rangle \right\|^2 + \sum_{j \in [i]} \left\| \left| \psi_{\text{bad}}^{(j)} \right\rangle \right\|^2 \tag{3.45}$$

$$= 1 \tag{3.46}$$

where the last step follows by the induction hypothesis. □

3.5 Faithful simulation of the good part

As any subtree of $\mathcal{G}$ without the EXIT vertex can be embedded in $\mathcal{T}$, one might expect that the size of the portion of the state $\left| \psi_{\mathcal{A}}^{(i)} \right\rangle$ that never encountered the EXIT or an almost cycle at any point in its history is the same as the size of the portion of the state $\left| \phi_{\mathcal{A}}^{(i)} \right\rangle$ that never encountered the EXIT or an almost cycle at any point in its history. We aim to formally argue this statement via a sequence of Lemmas that culminate in Lemma 54. We will restrict our attention to the good parts of the states $\left| \psi_{\mathcal{A}}^{(i)} \right\rangle$ and $\left| \phi_{\mathcal{A}}^{(i)} \right\rangle$ in this section, beginning with a useful decomposition of $\left| \phi_{\text{good}}^{(i)} \right\rangle$.

Definition 40. *We define an indexed expansion of* $\left| \phi_{\text{good}}^{(i)} \right\rangle$ *in the computational basis, as follows. Write* $\left| \phi_{\text{good}}^{(i)} \right\rangle = \sum_{p,q} \alpha_{p,q}^{(i)} \left| q^{(i)} \right\rangle \left| \phi_p^{(i)} \right\rangle$, *where each* $\left| \phi_p^{(i)} \right\rangle$ *denotes a computational basis state in the vertex register, each* $\left| q^{(i)} \right\rangle$ *specifies a computational basis state in the workspace register, and each* $\alpha_{p,q}^{(i)}$ *is an amplitude. Define* $\mathcal{P}_{\text{good}_{\mathcal{A}}}^{(i)}$ *to be the set of all indices* p *appearing in the expansion of* $\left| \phi_{\text{good}}^{(i)} \right\rangle$ *with any corresponding non-zero amplitude* $\alpha_{p,q}^{(i)}$.

We define computational basis states $\left| \psi_p^{(i)} \right\rangle$ from $\left| \phi_p^{(i)} \right\rangle$ and hence, from $\left| \phi_{\mathcal{A}}^{(i)} \right\rangle$ rather than $\left| \psi_{\mathcal{A}}^{(i)} \right\rangle$, which would be analogous to Definition 37.

Definition 41. *For $i \in [p(n)] \cup \{0\}$ and $p \in \mathcal{P}^{(i)}_{\text{good}_\mathcal{A}}$, let $\left|\psi_p^{(i)}\right\rangle := L\left|\phi_p^{(i)}\right\rangle$.*

Notice that it is not immediate by definition that $\left|\psi_p^{(i)}\right\rangle$ is in the support of the part of the state $\left|\psi_\mathcal{A}^{(i)}\right\rangle$ that is in the vertex space. However, by the end of this section, we will show that indeed this is the case.

For each $i \in [p(n)] \cup \{0\}$ and $p \in \mathcal{P}^{(i)}_{\text{good}_\mathcal{A}}$, the state $\left|\phi_p^{(i)}\right\rangle$ is a computational basis state in the address space, so we can write it as

$$\left|\phi_p^{(i)}\right\rangle = \bigotimes_{j \in [p(n)]} |s_j\rangle \tag{3.47}$$

for some strings $s_j \in \{0,1\}^{2p(n)}$.

Similarly, for each $i \in [p(n)] \cup \{0\}$ and $p \in \mathcal{P}^{(i)}_{\text{good}_\mathcal{A}}$, the state $\left|\psi_p^{(i)}\right\rangle$ is a computational basis state in the vertex space, so we can write it as

$$\left|\psi_p^{(i)}\right\rangle = \bigotimes_{j \in [p(n)]} |v_j\rangle \tag{3.48}$$

for some vertex labels $v_j \in \{0,1\}^{2n}$.

By the above notation and Definition 41, we have that for each $j \in [p(n)]$,

$$L(s_j) = v_j \tag{3.49}$$

Note that $|s_j\rangle$ and $|v_j\rangle$ also depend on p and i. However, as p and i will be clear from context, we keep this dependence implicit to simplify notation.

In the same vein, for each $i \in [p(n)] \cup \{0\}$ and workspace index q, the state $\left|q^{(i)}\right\rangle$ is a

computational basis states in the vertex space so we can write it as

$$\left|q^{(i)}\right\rangle = \bigotimes_{j\in[p(n)]} |w_j\rangle. \tag{3.50}$$

Again we suppress the dependence on q and i for simplicity.

Analogous to the notion of rooted state defined in Definition 11, we define the notion of address rooted state as follows. Informally, a state in the address space is address rooted if it contains a string that encodes an address, then it also contains the string that encodes its parent in $\mathcal{T}$.

Definition 42 (Address rooted state)**.** *We say that a computational basis state $|\phi\rangle$ in the address space is* address rooted *if for any string s stored in any of the registers of $|\phi\rangle$, whenever the vertex $B^{\mathsf{inv}}(s) \in \mathcal{V}_{\mathcal{T}}$ has a parent t, there exists a register of $|\phi\rangle$ that stores the string $B(t)$.*

Now, we will formally show that the notion of address rooted for states in the address space is analogous to the notion of rooted for states in the vertex space.

Lemma 43. *Let $i \in [p(n)] \cup \{0\}$, and $p \in \mathcal{P}^{(i)}_{\mathrm{good}_{\mathcal{A}}}$. If $\left|\psi_p^{(i)}\right\rangle$ is rooted, then $\left|\phi_p^{(i)}\right\rangle$ is address rooted.*

Proof. We prove the lemma by induction on i. By eq. (3.3), any register of the state $\left|\phi_p^{(0)}\right\rangle$ either stores EMPTYSTRING or $0^{2p(n)}$. We know, by Definition 24, that $B^{\mathsf{inv}}(\text{EMPTYSTRING}) = \text{EMPTYADDRESS}$ and $B^{\mathsf{inv}}(0^{2p(n)}) = \text{ZEROADDRESS}$ have no parents in the address tree $\mathcal{T}$. Hence, vacuously, $\left|\phi_p^{(0)}\right\rangle$ is address rooted.

Suppose, as the induction hypothesis, that the statement of the lemma is true for $i-1$ for some $i \geq 1$. Now, we show it holds for i.

If $\tilde{C}_i$ is not a controlled-oracle gate or $\tilde{C}_i$ is a controlled-oracle gate with the control workspace qubit in the $|0\rangle$ state, by Definition 28, either the positions of address registers of $\left|\phi_p\right\rangle$ change (and the positions of vertex registers of $\left|\psi_p\right\rangle$ change accordingly) or the contents of these address

(and vertex) registers do not change at all. In either case, by the induction hypothesis, the lemma follows.

Now, consider the case when $\tilde{C}_i = \wedge(\tilde{O}_c)$ for some $c \in \mathcal{C}$ and the corresponding control qubit in the workspace is in the $|0\rangle$ state. Any address register of $\left|\phi_{p'}^{(i-1)}\right\rangle$ that is not the target register of $\tilde{O}_c$ remain unchanged after $\tilde{O}_c$ is applied. This means that the corresponding vertex register of $\left|\psi_{p'}^{(i-1)}\right\rangle$ remain unchanged after $L\tilde{O}_c$ is applied. Thus, in order to argue that $\left|\phi_p^{(i)}\right\rangle$ is address rooted as in Definition 42, it is sufficient to show that for the string s is stored in the target register of $\tilde{O}_c$ in $\left|\phi_p^{(i)}\right\rangle$, if $B^{\mathsf{inv}}(s)$ has a parent t in $\mathcal{T}$, then $B(t)$ is stored in some register of $\left|\phi_p^{(i)}\right\rangle$, and if the target register of $\tilde{O}_c$ in $\left|\phi_{p'}^{(i-1)}\right\rangle$ stores some string $s^{(i-1)}$, then for any string s' stored in some register of $\left|\phi_{p'}^{(i-1)}\right\rangle$, $B^{\mathsf{inv}}(s^{(i-1)})$ cannot be the parent of $B^{\mathsf{inv}}(s')$.

Let $j \in [p(n)]$ be the index of the target register of $\left|\phi_p^{(i)}\right\rangle$. Recall from our convention in eqs. (3.47) and (3.48) that the jth registers of $\left|\phi_p^{(i)}\right\rangle$ and $\left|\psi_p^{(i)}\right\rangle$ store the states $|s_j\rangle$ and $|v_j\rangle$ respectively. Moreover, let $\left|s_j^{(i-1)}\right\rangle$ and $\left|v_j^{(i-1)}\right\rangle$ denote the states of the jth registers of $\tilde{O}_c^\dagger\left|\phi_p^{(i)}\right\rangle$ and $L\tilde{O}_c^\dagger\left|\phi_p^{(i)}\right\rangle$ respectively. Note that $\left|s_j^{(i-1)}\right\rangle$ is the state of jth register of $\left|\phi_{p'}^{(i-1)}\right\rangle$ for some $p' \in \mathcal{P}^{(i-1)}_{\mathrm{good}_{\mathcal{A}}}$.

Let k be such that the kth register, storing $|s_k\rangle$, of $\left|\phi_{p'}^{(i-1)}\right\rangle$ is the control address register for the gate O_c applied on $\left|\phi_{p'}^{(i-1)}\right\rangle$. Let $|v_k\rangle$ denote the state of the kth vertex register of $\left|\psi_{p'}^{(i-1)}\right\rangle$. From eq. (3.49), we know that $L(s_k) = v_k$ and $L(s_j^{(i-1)}) = v_j^{(i-1)}$. Let $t_k = B^{\mathsf{inv}}(s_k)$ and $t_j^{(i-1)} = B^{\mathsf{inv}}(s_j^{(i-1)})$. Then, $L'(t_k) = v_k$ and $L'(t_j^{(i-1)}) = v_j^{(i-1)}$. This means that t_k and $t_j^{(i-1)}$ are respective addresses of v_k and $v_j^{(i-1)}$ by Line 2. As per our discussion in Section 3.1, we have two cases.

1. $s_j^{(i-1)} = 0^{2p(n)}$. This means that $s_j = s_j^{(i-1)} \oplus B\lambda_c B^{\mathsf{inv}}(s_k) = B\lambda_c B^{\mathsf{inv}}(s_k)$.

   Notice that $t_j^{(i-1)} = B^{\mathsf{inv}}(s_j^{(i-1)}) = \textsc{zeroaddress}$ cannot be the parent of any vertex in $\mathcal{T}$ except for $\textsc{invalidaddress}$. But $\left|\phi_p^{(i)}\right\rangle$ cannot contain $B(\textsc{invalidaddress}) = \textsc{invalidstring}$ as otherwise, $\left|\psi_p^{(i)}\right\rangle$ would contain $\textsc{invalid}$ and so, will not be rooted by Definition 11,

contradicting the statement of the Lemma.

On the other hand, if $t_j = B^{\mathsf{inv}}(s_j) = \lambda_c B^{\mathsf{inv}}(s_k)$ has a parent in $\mathcal{T}$, then it will be either $B^{\mathsf{inv}}(s_k)$ or the grand parent of $B^{\mathsf{inv}}(s_k)$. In the former case, the parent of t_j is stored in the kth register of $\left|\phi_p^{(i)}\right\rangle$. In the latter case, t_j is the parent of $B^{\mathsf{inv}}(s_k)$. By the induction hypothesis, $B(t_j) = s_j$ is stored in some register of $\left|\phi_{p'}^{(i-1)}\right\rangle$. Let t_ℓ be the parent of t_j. Then, again by the induction hypothesis, $\left|\phi_{p'}^{(i-1)}\right\rangle$ stores $B(t_\ell)$ in its ℓth register for some $\ell \neq j$. Since $\tilde{O}_c$ only alters the content of its target register (i.e. jth register), the content of the ℓth register remain unchanged. It follows that $\left|\phi_p^{(i)}\right\rangle$ stores $B(t_\ell)$ in its ℓth register. Therefore, in this case, $\left|\phi_{p'}^{(i-1)}\right\rangle$ remains address rooted after $\tilde{O}_c$ has been applied. That is, $\left|\phi_p^{(i)}\right\rangle$ is address rooted.

2. $s_j^{(i-1)} = B\lambda_c B^{\mathsf{inv}}(s_k)$. This means that $s_j = s_j^{(i-1)} \oplus B\lambda_c B^{\mathsf{inv}}(s_k) = 0^{2p(n)}$.

   Since $s_j = 0^{2p(n)}$, $t_j = B^{\mathsf{inv}}(s_j) = \textsc{zeroaddress}$ does not have a parent in $\mathcal{T}$ by Definition 24.

   On the flip side, suppose that there is an $\ell \in [p(n)]$ such that $B^{\mathsf{inv}}(s_j^{(i-1)})$ is the parent of $B^{\mathsf{inv}}(s_\ell)$ where s_ℓ is the string stored in the ℓth register of $\left|\phi_{p'}^{(i-1)}\right\rangle$. Now consider the subgraphs $G_{\phi_{p'}}$ and G_{ϕ_p} of $\mathcal{G}$ induced by vertices in $\left|\psi_{p'}^{(i-1)}\right\rangle$ and $\left|\psi_p^{(i)}\right\rangle$. Since $p' \in \mathcal{P}_{\mathrm{good}_{\mathcal{A}}}^{(i-1)}$, we know that $G_{\phi_{p'}}$ cannot contain cycles. By the induction hypothesis, $\left|\phi_{p'}^{(i-1)}\right\rangle$ is address rooted. Thus, $\left|\psi_{p'}^{(i-1)}\right\rangle$ is rooted so $G_{\phi_{p'}}$ is connected. Therefore, $G_{\phi_{p'}}$ is a subtree of $\mathcal{G}$.

   We know, from eq. (3.49), that the vertex label v_ℓ stored in the ℓth register of $\left|\psi_{p'}^{(i-1)}\right\rangle$ is $L(s_\ell)$. Thus, by above, $v_j^{(i-1)}$ is the parent of v_ℓ in $G_{\phi_{p'}}$. That is, v_ℓ is connected to the ENTRANCE only through $v_j^{(i-1)}$ in $G_{\phi_{p'}}$. Recall that $\left|\psi_{p'}^{(i-1)}\right\rangle$ and $\left|\psi_p^{(i)}\right\rangle$ only differ on the jth register. This means that, as $v_j = 0^{2n}$, G_{ϕ_p} is $G_{\phi_{p'}}$ with $v_j^{(i-1)}$ (and the edges incident to it) removed. As a consequence, we have that v_ℓ is not connected to the ENTRANCE in G_{ϕ_p}. But $\left|\psi_p^{(i)}\right\rangle$ is rooted so this is not possible. It follows that $\left|\phi_p^{(i)}\right\rangle$ satisfies the sufficient

conditions of being address rooted stated above. □

As a consequence of Lemma 43, we show that the mapping L, defined in Definition 32, is a bijection from the set of address rooted states in $\left|\phi^{(i)}_{\text{good}}\right\rangle$ to the set of rooted states in $\left|\phi^{(i)}_{\text{good}}\right\rangle$ in the following Lemma.

Lemma 44. *Let $i \in [p(n)]$ and $p, p' \in \mathcal{P}^{(i)}_{\text{good}_\mathcal{A}}$. Suppose that $\left|\psi^{(i)}_p\right\rangle = \left|\psi^{(i)}_{p'}\right\rangle$ and $\left|\psi^{(i)}_p\right\rangle$ is rooted. Then $\left|\phi^{(i)}_p\right\rangle = \left|\phi^{(i)}_{p'}\right\rangle$.*

Proof. Suppose, towards contradiction, that $\left|\psi^{(i)}_p\right\rangle = \left|\psi^{(i)}_{p'}\right\rangle$ but $\left|\phi^{(i)}_p\right\rangle \neq \left|\phi^{(i)}_{p'}\right\rangle$. This means that there is an index $j \in [p(n)]$ such that the string s_j stored in the jth register of $\left|\phi^{(i)}_p\right\rangle$ is not equal to the string s'_j stored in the jth register of $\left|\phi^{(i)}_{p'}\right\rangle$, and yet the vertex label v_j stored in the jth register of $\left|\psi^{(i)}_p\right\rangle$ is equal to the vertex label v'_j stored in the jth register of $\left|\psi^{(i)}_{p'}\right\rangle$. Consider the addresses $t_j = B^{\text{inv}}(s_j)$ and $t'_j = B^{\text{inv}}(s'_j)$. We know that $L'(t_j) = L'(t'_j) = v_j$ from eq. (3.49).

By our discussion in section 3.1, we have s_j and s_k in the range of B. Recall, from Definition 25, that B is a bijection and $B^{\text{inv}} = B^{-1}$ on the range of B. Therefore, $t_j \neq t'_j$. Moreover, this means that we can write $s_j = B(t_j)$ and $s'_j = B(t'_j)$.

We know, from Lemma 43, that $\left|\phi^{(i)}_p\right\rangle$ and $\left|\phi^{(i)}_{p'}\right\rangle$ are address rooted as $\left|\psi^{(i)}_p\right\rangle$ and $\left|\psi^{(i)}_{p'}\right\rangle$ are rooted. This means that for any ancestor τ of t_j in $\mathcal{T}$, $B(\tau)$ is stored in one of the registers of $\left|\phi^{(i)}_p\right\rangle$. Therefore, there is a path from the vertex labeled EMPTYADDRESS and t_j in $\mathcal{T}$ such that $\left|\phi^{(i)}_p\right\rangle$ contains $B(\tau)$ for all vertices τ in this path. Let $\tau_0 = \text{EMPTYADDRESS}, \ldots, \tau_\gamma = t_j$ denote this path where γ denotes the length of this path. By Definition 41, $\left|\psi^{(i)}_p\right\rangle$ contains the vertex label $L'(\tau_i)$ for each $i \in [\gamma]$. Similarly, we can deduce that there is a path $\tau'_0 = \text{EMPTYADDRESS}, \ldots, \tau'_{\gamma'} = t'_j$ in $\mathcal{T}$, with γ' denoting the length of this path, such that $\left|\psi^{(i)}_{p'}\right\rangle = \left|\psi^{(i)}_p\right\rangle$ contains the vertex label $L'(\tau'_i)$ for each $i \in [\gamma']$. Since $t_j \neq t'_j$ and $v_j = v'_j$, it follows that $\left|\psi^{(i)}_p\right\rangle$ contains two distinct paths from $L'(\text{EMPTYADDRESS}) = \text{ENTRANCE}$ to $L'(t_j) = L'(t'_j) = v_j$ in $\mathcal{G}$. Hence, $\left|\psi^{(i)}_p\right\rangle$ contains

a cycle. But this is not possible since $p \in \mathcal{P}^{(i)}_{\text{good}_{\mathcal{A}}}$. Therefore, $\left|\phi^{(i)}_p\right\rangle \neq \left|\phi^{(i)}_{p'}\right\rangle \implies \left|\psi^{(i)}_p\right\rangle \neq \left|\psi^{(i)}_{p'}\right\rangle$. □

The following Lemma states that there is a bijective correspondence between the contents of the vertex registers of $\left|\psi^{(i)}_{\text{good}}\right\rangle$ and the address registers of $\left|\phi^{(i)}_{\text{good}}\right\rangle$.

Lemma 45. *Let $i \in [p(n)] \cup \{0\}$, $j,k \in [p(n)]$, $p \in \mathcal{P}^{(i)}_{\text{good}_{\mathcal{A}}}$ and $c \in \mathcal{C}$. Let $|v_j\rangle$ and $|v_k\rangle$ be the states stored in the jth and kth registers of $\left|\psi^{(i)}_p\right\rangle$ respectively as in eq. (3.48). Similarly, let $|s_j\rangle$ and $|s_k\rangle$ be the states stored in the jth and kth registers of $\left|\phi^{(i)}_p\right\rangle$ respectively as in eq. (3.47). Then*

1. $v_j = 0^{2n} \iff s_j = 0^{2p(n)}$,
2. $v_j = \text{NOEDGE} \iff s_j = \text{NOEDGESTRING}$,
3. $v_j = \text{INVALID} \iff s_j = \text{INVALIDSTRING}$,
4. $v_j = v_k \iff s_j = s_k$, *and*
5. $v_j = \eta_c(v_k) \iff s_j = B\lambda_c B^{\text{inv}}(s_k)$.

Proof. We show each statement separately as follows.

1. First, suppose that $v_j = 0^{2n}$. By eq. (3.49), we know that $L(s_j) = 0^{2n}$. Let $t_j = B^{\text{inv}}(s_j)$. Then, $L'(t_j) = 0^{2n}$. It can be observed from Line 2 that $t_j = \text{ZEROADDRESS}$ as only in that case $L'(t_j) = 0^{2n}$. Thus, by Definition 25, it must be that $s_j = 0^{2p(n)}$ as it is the only value of s_j for which $B^{\text{inv}}(s_j) = \text{ZEROADDRESS}$.

   Now, suppose that $s_j = 0^{2p(n)}$. Then,

$$v_j = L(s_j) = L'B^{\text{inv}}(0^{2p(n)}) = L'(\text{ZEROADDRESS}) = 0^{2n} \tag{3.51}$$

where the first step follows from Lemma 30, the third follows from Definition 25, and the fourth from Line 2.

2. First, suppose that $v_j = \text{NOEDGE}$. By eq. (3.49), we know that $L(s_j) = \text{NOEDGE}$. Let $t_j = B^{\text{inv}}(s_j)$. Then, $L'(t_j) = \text{NOEDGE}$. We claim that $t_j = \text{NOEDGEADDRESS}$. Indeed, if $t_j \neq \text{NOEDGE}$, then, as $L'(\text{NOEDGEADDRESS}) = L'(t_j) = \text{NOEDGE}$, we would have found the EXIT or a cycle in $\mathcal{G}$ by Lemma 30. But that is not possible since $p \in \mathcal{P}^{(i)}_{\text{good}_{\mathcal{A}}}$ so $t_j = \text{NOEDGEADDRESS}$. It follows that $s_j = \text{NOEDGESTRING}$ since it is the only possible assignment of s_j such that $B^{\text{inv}}(s_j) = \text{NOEDGEADDRESS}$ by Definition 25.

   Now, suppose that $s_j = \text{NOEDGE}$. Then,

$$v_j = L(s_j) = L'B^{\text{inv}}(\text{NOEDGESTRING}) = L'(\text{NOEDGEADDRESS}) = \text{NOEDGE} \tag{3.52}$$

   where the first step follows from Lemma 30, the third follows from Definition 25 and the fourth from Line 2.

3. First, suppose that $v_j = \text{INVALID}$. By eq. (3.49), we know that $L(s_j) = \text{INVALID}$. Let $t_j = B^{\text{inv}}(s_j)$. Then, $L'(t_j) = \text{INVALID}$. We claim that $t_j = \text{INVALIDADDRESS}$. Indeed, if $t_j \neq \text{INVALIDADDRESS}$, then, as $L'(\text{INVALIDADDRESS}) = L'(t_j) = \text{NOEDGE}$, we would have found the EXIT or a cycle in $\mathcal{G}$ by Lemma 30. But that is not possible since $p \in \mathcal{P}^{(i)}_{\text{good}_{\mathcal{A}}}$ so $t_j = \text{INVALIDADDRESS}$. Recall from section 3.1 that s_j is in the range of the B mapping in Definition 25. It follows that $s_j = \text{NOEDGESTRING}$ since it is the only possible assignment of s_j in the range of B with $B^{\text{inv}}(s_j) = \text{INVALIDADDRESS}$.

   Now, suppose that $s_j = \text{INVALID}$. Then,

$$v_j = L(s_j) = L'B^{\text{inv}}(\text{INVALIDSTRING}) = L'(\text{INVALIDADDRESS}) = \text{INVALID} \tag{3.53}$$

where the first step follows from Lemma 30, the third follows from Definition 25 and the fourth from Line 2.

4. First, suppose that $v_j = v_k$. From eq. (3.49), we can deduce that $L(s_j) = L(s_k)$. Let $t_j = B^{\text{inv}}(s_j)$ and $t_k = B^{\text{inv}}(s_k)$. Then, $L'(t_j) = L'(t_k)$. If $t_j \neq t_k$, then, by Lemma 30, the concatenation of the paths specified by the sequence of colors t_j and t_k forms a cycle, which contradicts $p \in \mathcal{P}^{(i)}_{\text{good}_\mathcal{A}}$. This means that $t_j = t_k$. Recall from section 3.1 that s_j and s_k in the range of B. Recall, from Definition 25, that B is a bijection and $B^{\text{inv}} = B^{-1}$ on the range of B. Therefore, $s_j = s_k$.

   Now suppose that $s_j = s_k$. Thus, $L(s_j) = L(s_k)$. By eq. (3.49), we know that $L(s_j) = v_j$ and $L(s_k) = v_k$. Therefore, $v_j = v_k$.

5. First, suppose that $v_j = \eta_c(v_k)$. From eq. (3.49), we have $v_j = L(s_j)$ and $v_k = L(s_k)$. Let $t_j = B^{\text{inv}}(s_j)$ and $t_k = B^{\text{inv}}(s_k)$. Then, $v_j = L'(t_j)$ and $v_k = L'(t_k)$, which means that t_j and t_k are addresses of the vertices v_j and v_k respectively. By Lemma 31, it follows that $\eta_c(v_k) = L'\lambda_c(t_k)$. Altogether, we have $L'(t_j) = L'\lambda_c(t_k)$. If $t_j \neq \lambda_c(t_k)$, then, by Lemma 30, the concatenation of the paths specified by the sequence of colors t_j and $\lambda_c(t_k)$ forms a cycle, which contradicts $p \in \mathcal{P}^{(i)}_{\text{good}_\mathcal{A}}$. This means that $t_j = \lambda_c(t_k)$. It follows that $BB^{\text{inv}}(s_j) = B(t_j) = B\lambda_c(t_k)$. Recall that s_j is in the range of B from section 3.1. Since B is a bijection and $B^{\text{inv}} = B^{-1}$ on the range of B, it follows that $BB^{\text{inv}}(s_j) = s_j$.

   Now suppose that $s_j = B\lambda_c B^{\text{inv}}(s_k)$. Thus, $L(s_j) = LB\lambda_c B^{\text{inv}}(s_k) = LB\lambda_c(t_k)$. By eq. (3.49) we have $v_j = LB\lambda_c(t_k)$. Note that $\lambda_c B^{\text{inv}}(s_k) = \lambda_c(t_k)$ is in the domain of B since B^{inv} maps any string to the domain of B and λ_c preserves the domain of B by Definitions 24 and 25. Therefore, as B is a bijection and $B^{\text{inv}} = B^{-1}$ on the range of B, $LB\lambda_c(t_k) = L'\lambda_c(t_k)$. Since t_k is an address of v_k, we have $L'\lambda_c(t_k) = \eta_c(v_k)$ by Lemma 31. Altogether, we

have $LB\lambda_c(t_k) = \eta_c(v_k)$. By the deductions we made above, we can conclude that $v_j = \eta_c(v_k)$. □

The next Lemma forms a key ingredient of Lemma 53, where we essentially show that the mapping L and the gate C_i commute: applying L followed by C_i is equivalent to applying $\tilde{C}_i$ followed by L.

Lemma 46. *Let $i \in [p(n)]$, $p \in \mathcal{P}^{(i)}_{\text{good}_\mathcal{A}}$ and q be any workspace index. Then, $L\tilde{C}_i\left|q^{(i-1)}\right\rangle\left|\phi_p^{(i-1)}\right\rangle = C_i\left|q^{(i-1)}\right\rangle\left|\psi_p^{(i-1)}\right\rangle$.*

Proof. We prove the statement of the Lemma for each of the possible gates in our gate set defined in Definition 9. For any quantum state $|\chi\rangle = \bigotimes_j |\chi_j\rangle$, and any indices $j_1, \ldots, j_r$, let $|\chi\rangle_{j_1,\ldots,j_r} \coloneqq |\chi_{j_1}\rangle \otimes \cdots \otimes |\chi_{j_r}\rangle$.

1. $C_i = \wedge(O_c)$ for some $c \in \mathcal{C}$. Let $x, y \in [p(n)]$ denote the vertex register indices that O_c acts on, and let a denote the workspace register index of the control qubit. Recall, from eqs. (3.47) and (3.48), that v_x and s_x denotes the contents of the xth vertex register of $\left|\psi_p^{(i-1)}\right\rangle$ and the xth address register of $\left|\phi_p^{(i-1)}\right\rangle$ respectively. Note that $t_x = B^{\text{inv}}(s_x)$ is an address of v_x as $L'(t_x) = v_x$ by eq. (3.49). Then,

$$LB\lambda_c B^{\text{inv}}(s_x) = L'BB^{\text{inv}}\lambda_c(t_x) = L'\lambda_c(t_x) = \eta_c(v_x) \tag{3.54}$$

where the first equality follows from Definition 32, the second follows from the bijectivity of B in Definition 25, and the third follows from Lemma 31.

Note that $|s_x\rangle$ and $|s_y\rangle$ denote the states of the control and target registers of the oracle gate $\tilde{O}_c$. We argued in section 3.1 that $\wedge(\tilde{O}_c)$ is controlled on an address workspace register which is 1 only if $s_y = 0^{2p(n)}$ or $s_y = B\lambda_{c^*}B^{\text{inv}}(s_x)$, which is based on an assumption we made without loss of generality in section 2.2.

$$L \wedge (\tilde{O}_c) \left| \phi_p^{(i-1)} \right\rangle_{x,y} \left| q^{(i-1)} \right\rangle_a = L \wedge (\tilde{O}_c) |s_x\rangle |s_y\rangle |w_a\rangle \tag{3.55}$$

$$= L \begin{cases} |s_x\rangle |s_y \oplus B\lambda_c B^{\mathsf{inv}}(s_x)\rangle |w_a\rangle & w_a = 1 \\ |s_x\rangle |s_y\rangle |w_a\rangle & \text{otherwise} \end{cases} \tag{3.56}$$

$$= L \begin{cases} |s_x\rangle \left| B\lambda_c B^{\mathsf{inv}}(s_x) \right\rangle |w_a\rangle & s_y = 0^{2p(n)} \;\&\; w_a = 1 \\ |s_x\rangle \left| 0^{2p(n)} \right\rangle |w_a\rangle & s_y = B\lambda_{c^*} B^{\mathsf{inv}}(s_x) \;\&\; w_a = 1 \\ |s_x\rangle |s_y\rangle |w_a\rangle & \text{otherwise} \end{cases} \tag{3.57}$$

$$= \begin{cases} |v_x\rangle |\eta_c(v_x)\rangle |w_a\rangle & v_y = 0^{2n} \;\&\; w_a = 1 \\ |v_x\rangle |0^{2n}\rangle |w_a\rangle & v_y = \eta_c(v_x) \;\&\; w_a = 1 \\ |v_x\rangle |v_y\rangle |w_a\rangle & \text{otherwise} \end{cases} \tag{3.58}$$

$$= |v_x\rangle |v_y \oplus w_a \cdot \eta_c(v_x)\rangle |w_a\rangle \tag{3.59}$$

$$= \wedge(O_c) |v_x\rangle |v_y\rangle |w_a\rangle \tag{3.60}$$

$$= \wedge(O_c) \left| \psi_p^{(i-1)} \right\rangle_{x,y} \left| q^{(i-1)} \right\rangle_a \tag{3.61}$$

where the second from Definition 28; the third from an observation made above; the fourth from eq. (3.49), parts 1 and 5 of Lemma 45, and eq. (3.54); and the sixth from Definition 9.

2. $C_i = \wedge(e^{i\theta T})$ for some $\theta \in [0, 2\pi)$. Let $x, y \in [p(n)]$ denote the vertex register indices that

$e^{i\theta T}$ acts on, and let a denote the workspace register index of the control qubit. Then

$$L\wedge(e^{i\theta\tilde{T}})\left|\phi_p^{(i-1)}\right\rangle_{x,y}\left|q^{(i-1)}\right\rangle_a = L\wedge(e^{i\theta\tilde{T}})|s_x\rangle|s_y\rangle|w_a\rangle \tag{3.62}$$

$$= L\begin{cases}\cos\theta|s_x\rangle|s_y\rangle|w_a\rangle + i\sin\theta|s_y\rangle|s_x\rangle|w_a\rangle & w_a = 1\\ |s_x\rangle|s_y\rangle|w_a\rangle & \text{otherwise}\end{cases} \tag{3.63}$$

$$= \begin{cases}\cos\theta|v_x\rangle|v_y\rangle|w_a\rangle + i\sin\theta|v_y\rangle|v_x\rangle|w_a\rangle & w_a = 1\\ |v_x\rangle|v_y\rangle|w_a\rangle & \text{otherwise}\end{cases} \tag{3.64}$$

$$= \wedge(e^{i\theta T})|v_x\rangle|v_y\rangle|w_a\rangle \tag{3.65}$$

$$= \wedge(e^{i\theta T})\left|\psi_p^{(i-1)}\right\rangle_{x,y}\left|q^{(i-1)}\right\rangle_a \tag{3.66}$$

where the second from Definition 28, the third from eq. (3.49), and the fourth from Definition 9.

3. $C_i = \mathcal{E}$. Let $x, y \in [p(n)]$ denote the vertex register indices and a denote the workspace register index that $\mathcal{E}$ acts on. Then,

$$L\tilde{\mathcal{E}}\left|\phi_p^{(i-1)}\right\rangle_{x,y}\left|q^{(i-1)}\right\rangle_a = L\tilde{\mathcal{E}}|s_x\rangle|s_y\rangle|w_a\rangle \tag{3.67}$$

$$= L\big(|s_x\rangle|s_y\rangle\big)|w_a \oplus \delta[s_x = s_y]\rangle \tag{3.68}$$

$$= |v_x\rangle|v_y\rangle|w_a \oplus \delta[v_x = v_y]\rangle \tag{3.69}$$

$$= \mathcal{E}|v_x\rangle|v_y\rangle|w_a\rangle \tag{3.70}$$

$$= \mathcal{E}\left|\psi_p^{(i-1)}\right\rangle_{x,y}\left|q^{(i-1)}\right\rangle_a \tag{3.71}$$

where the second from Definition 28, the third from eq. (3.49) and part 4 of Lemma 45, and the fourth from Definition 9.

4. $C_i = \mathcal{N}$. Let $x \in [p(n)]$ denote the vertex register index and a denote the workspace register index that $\mathcal{N}$ acts on. Then,

$$L\tilde{\mathcal{N}}\left|\phi_p^{(i-1)}\right\rangle_x \left|q^{(i-1)}\right\rangle_a = L\tilde{\mathcal{N}}|s_x\rangle|w_a\rangle \tag{3.72}$$

$$= L(|s_x\rangle)|w_a \oplus \delta[s_x = \text{NOEDGESTRING}]\rangle \tag{3.73}$$

$$= |v_x\rangle|w_a \oplus \delta[v_x = \text{NOEDGE}]\rangle \tag{3.74}$$

$$= \mathcal{N}|v_x\rangle|w_a\rangle \tag{3.75}$$

$$= \mathcal{N}\left|\psi_p^{(i-1)}\right\rangle_x \left|q^{(i-1)}\right\rangle_a \tag{3.76}$$

where the second from Definition 28, the third from eq. (3.49) and part 2 of Lemma 45, and the fourth from Definition 9.

5. $C_i = \mathcal{Z}$. Let $x \in [p(n)]$ denote the vertex register index and a denote the workspace register index that $\mathcal{N}$ acts on. Then,

$$L\tilde{\mathcal{Z}}\left|\phi_p^{(i-1)}\right\rangle_x \left|q^{(i-1)}\right\rangle_a = L\tilde{\mathcal{N}}|s_x\rangle|w_a\rangle \tag{3.77}$$

$$= L(|s_x\rangle)\left|w_a \oplus \delta[s_x = 0^{2p(n)}]\right\rangle \tag{3.78}$$

$$= |v_x\rangle\left|w_a \oplus \delta[v_x = 0^{2n}]\right\rangle \tag{3.79}$$

$$= \mathcal{Z}|v_x\rangle|w_a\rangle \tag{3.80}$$

$$= \mathcal{Z}\left|\psi_p^{(i-1)}\right\rangle_x \left|q^{(i-1)}\right\rangle_a \tag{3.81}$$

where the second from Definition 28, the third from eq. (3.49) and part 1 of Lemma 45, and

the fourth from Definition 9.

6. C_i is a gate on the workspace register. Since L acts on the address space, L and C_i commute. Moreover, $C_i = \tilde{C}_i$ as we do not replace the gates acting on the workspace register in Definition 28. Thus,

$$L\tilde{C}_i\left|\phi_p^{(i-1)}\right\rangle\left|q^{(i-1)}\right\rangle = \tilde{C}_i L\left|\phi_p^{(i-1)}\right\rangle\left|q^{(i-1)}\right\rangle = C_i\left|\psi_p^{(i-1)}\right\rangle\left|q^{(i-1)}\right\rangle. \tag{3.82}$$

□

Notice that the non-oracle gates in Definition 9 does not produce any 'new information' about vertex labels. Based on this intuition, one might expect that the portion of $\left|\psi_{\mathcal{A}}^{(i-1)}\right\rangle$ (respectively $\left|\phi_{\mathcal{A}}^{(i-1)}\right\rangle$) that has never encountered the EXIT or a cycle will not encounter the EXIT or a cycle on the application of C_i (respectively $\tilde{C}_i$) at the ith step. We formalize this expectation as follows.

Lemma 47. *Let $i \in [p(n)]$ and suppose that C_i is a genuine non-oracle gate. Then*

1. $\left|\phi_{\text{good}}^{(i)}\right\rangle = \tilde{C}_i\left|\phi_{\text{good}}^{(i-1)}\right\rangle$ *and*
2. $\left|\psi_{\text{good}}^{(i)}\right\rangle = C_i\left|\psi_{\text{good}}^{(i-1)}\right\rangle$.

Proof. We show part 1 using part 7 of Lemma 38. Part 2 follows by an analogous argument that instead uses part 8 of Lemma 38.

Notice that in Definition 9, the only gates that alter the vertex space are O_c gates or $e^{i\theta T}$ gates. But the $e^{i\theta T}$ gates only swaps contents of the vertex register (without computing a new vertex label in a vertex register). In other words, genuine non-oracle gates does not introduce new vertex labels. This means that, as is the case with $\left|\phi_{\text{good}}^{(i-1)}\right\rangle$, the subgraph corresponding to any computational basis state in the support of $\tilde{C}_i\left|\phi_{\text{good}}^{(i)}\right\rangle$ will not contain the EXIT and will be

more than one edge away from containing a cycle. That is, $\left|\phi^{(i)}_{\text{bad}}\right\rangle = 0$. Therefore, by part 7 of Lemma 38, we have $C_i\left|\phi^{(i-1)}_{\text{good}}\right\rangle = \left|\phi^{(i)}_{\text{good}}\right\rangle + \left|\phi^{(i)}_{\text{bad}}\right\rangle = \left|\phi^{(i)}_{\text{good}}\right\rangle$. □

For the analysis of oracle gates, we now define a subset of $\mathcal{P}^{(i-1)}_{\text{good}_\mathcal{A}}$ which contains indices corresponding to computational basis states in the address space that does not contain the EXIT or a cycle even after the application of an oracle gate at the ith step. Inspired by the decomposition in Definition 40, we then define the components $\left|\phi^{(i-1)}_{\text{great}}\right\rangle$ and $\left|\psi^{(i-1)}_{\text{great}}\right\rangle$ of $\left|\phi^{(i-1)}_{\mathcal{A}}\right\rangle$ and $\left|\psi^{(i-1)}_{\mathcal{A}}\right\rangle$ respectively.

Definition 48. *Let $i \in [p(n)]$. Suppose that $C_i = \wedge(O_c)$ for some $c \in \mathcal{C}$. Then, define*

$$\mathcal{P}^{(i-1)}_{\text{great}_\mathcal{A}} := \left\{ p \in \mathcal{P}^{(i-1)}_{\text{good}_\mathcal{A}} : \exists p' \in \mathcal{P}^{(i)}_{\text{good}_\mathcal{A}} \text{ such that } \tilde{C}_i\left|\phi^{(i-1)}_p\right\rangle = \left|\phi^{(i)}_{p'}\right\rangle \right\}. \tag{3.83}$$

Also, let

$$\left|\phi^{(i-1)}_{\text{great}}\right\rangle := \sum_{p\in\mathcal{P}^{(i-1)}_{\text{great}_\mathcal{A}}} \sum_q \alpha^{(i-1)}_{p,q}\left|q^{(i-1)}\right\rangle\left|\phi^{(i-1)}_p\right\rangle, \qquad \left|\psi^{(i-1)}_{\text{great}}\right\rangle := \sum_{p\in\mathcal{P}^{(i-1)}_{\text{great}_\mathcal{A}}} \sum_q \alpha^{(i-1)}_{p,q}\left|q^{(i-1)}\right\rangle\left|\psi^{(i-1)}_p\right\rangle. \tag{3.84}$$

In the following Lemma, we show that the L mapping preserves the relationship between computational basis states in the support of $\left|\phi^{(i-1)}_{\text{good}}\right\rangle$ and $\left|\phi^{(i)}_{\text{good}}\right\rangle$: applying the oracle gate to a computational basis state in the support of $\left|\phi^{(i-1)}_{\text{good}}\right\rangle$ will result in a computational basis state in the support of $\left|\phi^{(i)}_{\text{good}}\right\rangle$ exactly when applying the oracle gate to a computational basis state in the support of $L\left|\phi^{(i-1)}_{\text{good}}\right\rangle$ results in a computational basis state in the support of $L\left|\phi^{(i)}_{\text{good}}\right\rangle$.

Lemma 49. *Let $i \in [p(n)]$, $p \in \mathcal{P}^{(i-1)}_{\text{good}_\mathcal{A}}$ and $p' \in \mathcal{P}^{(i)}_{\text{good}_\mathcal{A}}$. Suppose that $C_i = \wedge(O_c)$ for some $c \in \mathcal{C}$ and $L\left|\phi^{(i-1)}_{\text{good}}\right\rangle = \left|\psi^{(i-1)}_{\text{good}}\right\rangle$. Then, $\tilde{C}_i\left|\phi^{(i-1)}_p\right\rangle = \left|\phi^{(i)}_{p'}\right\rangle$ iff $C_i\left|\psi^{(i-1)}_p\right\rangle = \left|\psi^{(i)}_{p'}\right\rangle$.*

Proof. First, suppose that $\tilde{C}_i \left|\phi_p^{(i-1)}\right\rangle = \left|\phi_{p'}^{(i)}\right\rangle$. Then

$$C_i \left|\psi_p^{(i-1)}\right\rangle = L\tilde{C}_i \left|\phi_p^{(i-1)}\right\rangle = L\left|\phi_{p'}^{(i)}\right\rangle = \left|\psi_{p'}^{(i)}\right\rangle \tag{3.85}$$

where we used part 1 of Lemma 46 in the first step, the assumption made above in the second, and Definition 41 in the last step.

Now, suppose that $C_i \left|\psi_p^{(i-1)}\right\rangle = \left|\psi_{p'}^{(i)}\right\rangle$. By the statement of the Lemma, we have that $\tilde{C}_i = \tilde{O}_c$, which means that $\tilde{C}_i$ is a permutation of the computational basis states. Thus, since $\left|\phi_p^{(i-1)}\right\rangle$ is in the support of $\left|\phi_{\text{good}}^{(i-1)}\right\rangle$, we will have $\tilde{C}_i \left|\phi_p^{(i-1)}\right\rangle$ in the support of $\tilde{C}_i \left|\phi_{\text{good}}^{(i-1)}\right\rangle$. From part 7 of Lemma 38, we know that $\tilde{C}_i \left|\phi_{\text{good}}^{(i-1)}\right\rangle = \left|\phi_{\text{good}}^{(i)}\right\rangle + \left|\phi_{\text{bad}}^{(i)}\right\rangle$. But since $p' \in \mathcal{P}^{(i)}_{\text{good}_\mathcal{A}}$ and $\tilde{C}_i \left|\phi_p^{(i-1)}\right\rangle = \left|\phi_{p'}^{(i)}\right\rangle$, it is not possible for the computational basis state $\tilde{C}_i \left|\phi_p^{(i-1)}\right\rangle$ to be ϕ-bad. In other words, $\Pi^{\phi}_{\text{bad}} \tilde{C}_i \left|\phi_p^{(i-1)}\right\rangle = 0$. Recall that $\Pi^{\phi}_{\text{bad}} \left|\phi_{\text{bad}}^{(i)}\right\rangle = \left|\phi_{\text{bad}}^{(i)}\right\rangle$ from part 1 of Lemma 38. Thus, $\tilde{C}_i \left|\phi_p^{(i-1)}\right\rangle$ cannot be in the support of $\left|\phi_{\text{bad}}^{(i)}\right\rangle$, which means that $\tilde{C}_i \left|\phi_p^{(i-1)}\right\rangle$ is in the support of $\left|\phi_{\text{good}}^{(i)}\right\rangle$. This means that there is a $p'' \in \mathcal{P}^{(i)}_{\text{good}_\mathcal{A}}$ such that $\left|\phi_{p''}^{(i)}\right\rangle = \tilde{C}_i \left|\phi_p^{(i-1)}\right\rangle$. Thus, $\left|\psi_{p''}^{(i)}\right\rangle = L\tilde{C}_i \left|\phi_p^{(i-1)}\right\rangle$.

By a similar argument to the one we made just above and the assumption that $L\left|\phi_{\text{good}}^{(i-1)}\right\rangle = \left|\psi_{\text{good}}^{(i-1)}\right\rangle$, we can see that $C_i \left|\psi_p^{(i-1)}\right\rangle$ is in the support of $\left|\psi_{\text{good}}^{(i)}\right\rangle$. Therefore, $C_i \left|\psi_p^{(i-1)}\right\rangle$ is a rooted state by our discussion following Lemma 38. By Lemma 46, we can deduce that $L\tilde{C}_i \left|\phi_p^{(i-1)}\right\rangle = C_i \left|\psi_p^{(i-1)}\right\rangle$, which implies that $\left|\psi_{p''}^{(i)}\right\rangle = C_i \left|\psi_p^{(i-1)}\right\rangle$. Since $\left|\psi_{p'}^{(i)}\right\rangle = C_i \left|\psi_p^{(i-1)}\right\rangle = \left|\psi_{p''}^{(i)}\right\rangle$ and $C_i \left|\psi_p^{(i-1)}\right\rangle$ is rooted, we get that $\left|\phi_{p'}^{(i)}\right\rangle = \left|\phi_{p''}^{(i)}\right\rangle$ from Lemma 44. Hence, $\left|\phi_{p'}^{(i)}\right\rangle = \tilde{C}_i \left|\phi_p^{(i-1)}\right\rangle$. □

Lemma 49 and Definition 48 gives rise to an alternative definition of $\mathcal{P}^{(i-1)}_{\text{great}_\mathcal{A}}$, which is given by the following corollary.

Corollary 50. *Let $i \in p[n]$. Suppose that $C_i = \wedge(O_c)$ for some $c \in \mathcal{C}$ and $L\left|\phi^{(i-1)}_{\text{good}}\right\rangle = \left|\psi^{(i-1)}_{\text{good}}\right\rangle$. Then,*

$$\mathcal{P}^{(i-1)}_{\text{great}_{\mathcal{A}}} = \left\{ p \in \mathcal{P}^{(i-1)}_{\text{good}_{\mathcal{A}}} : \exists p' \in \mathcal{P}^{(i)}_{\text{good}_{\mathcal{A}}} \text{ such that } C_i\left|\psi^{(i-1)}_{p}\right\rangle = \left|\psi^{(i)}_{p'}\right\rangle \right\}. \tag{3.86}$$

The next Lemma provides two equivalent conditions for membership in $\mathcal{P}^{(i-1)}_{\text{great}_{\mathcal{A}}}$: for any index $p \in \mathcal{P}^{(i-1)}_{\text{good}_{\mathcal{A}}}$, we have $p \in \mathcal{P}^{(i-1)}_{\text{great}_{\mathcal{A}}}$ exactly when the oracle gate at the ith step applied to the computational basis state in the address space (respectively vertex space) associated with p results in a computational basis state in the support of $\left|\phi^{(i)}_{\text{good}}\right\rangle$ $\left(\text{respectively } \left|\psi^{(i)}_{\text{good}}\right\rangle\right)$.

Lemma 51. *Let $i \in [p(n)]$, $p \in \mathcal{P}^{(i-1)}_{\text{good}_{\mathcal{A}}}$. Suppose that $C_i = \wedge(O_c)$ for some $c \in \mathcal{C}$ and $L\left|\phi^{(i-1)}_{\text{good}}\right\rangle = \left|\psi^{(i-1)}_{\text{good}}\right\rangle$. Then*

1. *$p \in \mathcal{P}^{(i-1)}_{\text{great}_{\mathcal{A}}}$ iff $\left\langle \phi^{(i-1)}_{p}\right|\tilde{C}_i\left|\phi^{(i)}_{\text{good}}\right\rangle \neq 0$ and*
2. *$p \in \mathcal{P}^{(i-1)}_{\text{great}_{\mathcal{A}}}$ iff $\left\langle \psi^{(i-1)}_{p}\right|C_i\left|\psi^{(i)}_{\text{good}}\right\rangle \neq 0$.*

Proof. We prove part 1 using Definitions 40 and 48. Part 2 follows by an analogous argument that, instead, uses the alternative definition of $\mathcal{P}^{(i-1)}_{\text{great}_{\mathcal{A}}}$ in Corollary 50, and the assumption $L\left|\phi^{(i-1)}_{\text{good}}\right\rangle = \left|\psi^{(i-1)}_{\text{good}}\right\rangle$ along with Definition 40.

First, suppose that $p \in \mathcal{P}^{(i-1)}_{\text{great}_{\mathcal{A}}}$. Then, by Definition 48, there is some $p' \in \mathcal{P}^{(i)}_{\text{good}_{\mathcal{A}}}$ such that $\tilde{C}_i\left|\phi^{(i-1)}_{p}\right\rangle = \left|\phi^{(i)}_{p'}\right\rangle$. It follows that $\left\langle \phi^{(i-1)}_{p}\right|\tilde{C}_i\left|\phi^{(i)}_{\text{good}}\right\rangle = \left\langle \phi^{(i)}_{p'}\right|\left|\phi^{(i)}_{\text{good}}\right\rangle$. Since $p' \in \mathcal{P}^{(i)}_{\text{good}_{\mathcal{A}}}$, we can deduce that $\left\langle \phi^{(i)}_{p'}\right|\left|\phi^{(i)}_{\text{good}}\right\rangle \neq 0$ by Definition 40.

Conversely, suppose that $p \notin \mathcal{P}^{(i-1)}_{\text{great}_{\mathcal{A}}}$. Then, by Definition 48, $\tilde{C}_i\left|\phi^{(i-1)}_{p}\right\rangle \neq \left|\phi^{(i)}_{p'}\right\rangle$ for all $p' \in \mathcal{P}^{(i)}_{\text{good}_{\mathcal{A}}}$. Since $\left|\phi^{(i)}_{p'}\right\rangle$ and $\tilde{C}_i\left|\phi^{(i-1)}_{p}\right\rangle$ are computational basis states (in the vertex space), it means that $\left\langle \phi^{(i-1)}_{p}\right|\tilde{C}_i\left|\phi^{(i)}_{p'}\right\rangle = 0$ for all $p' \in \mathcal{P}^{(i)}_{\text{good}_{\mathcal{A}}}$. By Definition 40, we know that $\left|\phi^{(i)}_{\text{good}}\right\rangle$ is supported only on states in $\left\{\left|\phi^{(i)}_{p'}\right\rangle : p' \in \mathcal{P}^{(i)}_{\text{good}_{\mathcal{A}}}\right\}$. It follows that $\left\langle \phi^{(i-1)}_{p}\right|\tilde{C}_i\left|\phi^{(i)}_{\text{good}}\right\rangle = 0$. □

By the definition of $\mathcal{P}^{(i-1)}_{\text{great}_{\mathcal{A}}}$, there seems to be a bijective correspondence between elements

of $\mathcal{P}^{(i-1)}_{\text{great}_{\mathcal{A}}}$ and $\mathcal{P}^{(i)}_{\text{good}_{\mathcal{A}}}$. We make this precise by showing that the application of the oracle gate at the ith step to the state $\left|\phi^{(i-1)}_{\text{great}}\right\rangle$ (respectively $\left|\psi^{(i-1)}_{\text{great}}\right\rangle$), defined in Definition 48, results in the state $\left|\phi^{(i)}_{\text{good}}\right\rangle$ (respectively $\left|\psi^{(i)}_{\text{good}}\right\rangle$) using Lemma 51.

Lemma 52. *Let $i \in [p(n)]$ and $c \in \mathcal{C}$. Suppose that $C_i = \wedge(O_c)$ for some $c \in \mathcal{C}$ and $L\left|\phi^{(i-1)}_{\text{good}}\right\rangle = \left|\psi^{(i-1)}_{\text{good}}\right\rangle$. Then*

1. $\left|\phi^{(i)}_{\text{good}}\right\rangle = \tilde{C}_i\left|\phi^{(i-1)}_{\text{great}}\right\rangle$ *and*
2. $\left|\psi^{(i)}_{\text{good}}\right\rangle = C_i\left|\psi^{(i-1)}_{\text{great}}\right\rangle$.

Proof. We show part 1 using part 7 of Lemma 38, part 1 of Lemma 51, and Definition 40. Part 2 follows by an analogous argument that, instead, uses part 8 of Lemma 38, part 2 of Lemma 51, and the assumption $L\left|\phi^{(i-1)}_{\text{good}}\right\rangle = \left|\psi^{(i-1)}_{\text{good}}\right\rangle$ along with Definition 40.

Expanding $\tilde{C}_i\left|\phi^{(i-1)}_{\text{good}}\right\rangle$ using Definitions 40 and 48 results in

$$\tilde{C}_i\left|\phi^{(i-1)}_{\text{good}}\right\rangle = \sum_{p\in\mathcal{P}^{(i-1)}_{\text{great}_{\mathcal{A}}}}\sum_q \alpha^{(i-1)}_{p,q}\left|q^{(i-1)}\right\rangle\tilde{C}_i\left|\phi^{(i-1)}_p\right\rangle + \sum_{\substack{p\in\mathcal{P}^{(i-1)}_{\text{good}_{\mathcal{A}}}\\ p\notin\mathcal{P}^{(i-1)}_{\text{great}_{\mathcal{A}}}}}\sum_q \alpha^{(i-1)}_{p,q}\left|q^{(i-1)}\right\rangle\tilde{C}_i\left|\phi^{(i-1)}_p\right\rangle. \tag{3.87}$$

Combining it with part 7 of Lemma 38, we get

$$\left|\phi^{(i)}_{\text{good}}\right\rangle + \left|\phi^{(i)}_{\text{bad}}\right\rangle = \sum_{p\in\mathcal{P}^{(i-1)}_{\text{great}_{\mathcal{A}}}}\sum_q \alpha^{(i-1)}_{p,q}\left|q^{(i-1)}\right\rangle\tilde{C}_i\left|\phi^{(i-1)}_p\right\rangle + \sum_{\substack{p\in\mathcal{P}^{(i-1)}_{\text{good}_{\mathcal{A}}}\\ p\notin\mathcal{P}^{(i-1)}_{\text{great}_{\mathcal{A}}}}}\sum_q \alpha^{(i-1)}_{p,q}\left|q^{(i-1)}\right\rangle\tilde{C}_i\left|\phi^{(i-1)}_p\right\rangle. \tag{3.88}$$

Let $p \in \mathcal{P}^{(i-1)}_{\text{great}_{\mathcal{A}}}$. Then, by part 1 of Lemma 51, $\left\langle\phi^{(i)}_{\text{good}}\right|\tilde{C}_i\left|\phi^{(i-1)}_p\right\rangle \neq 0$. Recall from part 1 of Lemma 38 that $\left|\phi^{(i)}_{\text{good}}\right\rangle$ and $\left|\phi^{(i)}_{\text{bad}}\right\rangle$ have disjoint support. Since $\tilde{C}_i\left|\phi^{(i-1)}_p\right\rangle$ is a computational basis state, this means that $\left\langle\phi^{(i)}_{\text{bad}}\right|\tilde{C}_i\left|\phi^{(i-1)}_p\right\rangle = 0$. Now, let $p' \in \mathcal{P}^{(i-1)}_{\text{good}_{\mathcal{A}}}$. Then, as $\left|\phi^{(i-1)}_p\right\rangle$ and $\left|\phi^{(i-1)}_{p'}\right\rangle$ are computational basis states, $\left\langle\phi^{(i-1)}_{p'}\right|\tilde{C}_i^\dagger\tilde{C}_i\left|\phi^{(i-1)}_p\right\rangle = \left\langle\phi^{(i-1)}_{p'}\middle|\phi^{(i-1)}_p\right\rangle$ is 1 when $p = p'$

and 0 otherwise. Therefore, we can deduce from eq. (3.88), that

$$\left\langle \phi_p^{(i-1)} \middle| \tilde{C}_i^\dagger \middle| \phi_{\text{good}}^{(i)} \right\rangle = \sum_q \alpha_{p,q}^{(i-1)} \left| q^{(i-1)} \right\rangle. \tag{3.89}$$

For any $p \notin \mathcal{P}^{(i-1)}_{\text{great}_{\mathcal{A}}}$, $\left\langle \phi_p^{(i-1)} \middle| \tilde{C}_i^\dagger \middle| \phi_{\text{good}}^{(i)} \right\rangle = 0$ by part 1 of Lemma 51. Combining it with eq. (3.89), we can conclude that

$$\left| \phi_{\text{good}}^{(i)} \right\rangle = \sum_{p \in \mathcal{P}^{(i-1)}_{\text{great}_{\mathcal{A}}}} \sum_q \alpha_{p,q}^{(i-1)} \left| q^{(i-1)} \right\rangle \tilde{C}_i \left| \phi_p^{(i-1)} \right\rangle = \tilde{C}_i \left| \phi_{\text{good}}^{(i-1)} \right\rangle, \tag{3.90}$$

where the last equality follows by Definition 48. □

Much of the above analysis in this section will help us establish the following Lemma, which states that the states $\left| \phi_{\text{good}}^{(i)} \right\rangle$ and $\left| \psi_{\text{good}}^{(i)} \right\rangle$ are related by the mapping L. Intuitively, the oracle $\tilde{O}$ based on the address tree $\mathcal{T}$ can faithfully simulate (modulo mapping L) the portion of the state $\left| \psi_{\mathcal{A}}^{(i)} \right\rangle$ of the algorithm $\mathcal{A}$ that does not encounter the EXIT or a cycle.

Lemma 53. *For all $i \in [p(n)] \cup \{0\}$, $L \left| \phi_{\text{good}}^{(i)} \right\rangle = \left| \psi_{\text{good}}^{(i)} \right\rangle$.*

Proof. We prove this Lemma by induction on i.

For $i = 0$, note that

$$\left| \phi_{\text{good}}^{(0)} \right\rangle = \left| \phi_{\mathcal{A}}^{(0)} \right\rangle - \left| \phi_{\text{allbad}}^{(0)} \right\rangle = \left| \phi_{\mathcal{A}}^{(0)} \right\rangle = |\text{EMPTYSTRING}\rangle \otimes \left| 0^{2p(n)} \right\rangle^{\otimes (p(n)-1)} \otimes |0\rangle_{\text{workspace}} \tag{3.91}$$

where the first equality follows from part 5 of Lemma 38, the second equality follows from Definition 37, and the last equality follows from Definition 28. Similarly,

$$\left| \psi_{\text{good}}^{(0)} \right\rangle = \left| \psi_{\mathcal{A}}^{(0)} \right\rangle - \left| \psi_{\text{allbad}}^{(0)} \right\rangle = \left| \psi_{\mathcal{A}}^{(0)} \right\rangle = |\text{ENTRANCE}\rangle \otimes \left| 0^{2n} \right\rangle^{\otimes (p(n)-1)} \otimes |0\rangle_{\text{workspace}} \tag{3.92}$$

where the first equality follows from part 6 of Lemma 38, the second equality follows from Definition 37, and the last equality follows from Definition 9. The statement of the lemma for $i = 0$ follows by noticing that $L(\text{EMPTYSTRING}) = \text{ENTRANCE}$ and $L(0^{2n}) = 0^{2p(n)}$ from eq. (3.49), respectively.

Now, assume that $L\left|\phi_{\text{good}}^{(i-1)}\right\rangle = \left|\psi_{\text{good}}^{(i-1)}\right\rangle$ for some $i \in [p(n)]$. Then, we have two cases depending on C_i:

1. C_i is a genuine non-oracle gate. Then

$$L\left|\phi_{\text{good}}^{(i)}\right\rangle = L\tilde{C}_i\left|\phi_{\text{good}}^{(i-1)}\right\rangle \tag{3.93}$$

$$= \sum_{p \in \mathcal{P}_{\text{good}_{\mathcal{A}}}^{(i-1)}} \sum_q \alpha_{p,q}^{(i-1)} L\tilde{C}_i\left|q^{(i-1)}\right\rangle\left|\phi_p^{(i-1)}\right\rangle \tag{3.94}$$

$$= \sum_{p \in \mathcal{P}_{\text{good}_{\mathcal{A}}}^{(i-1)}} \sum_q \alpha_{p,q}^{(i-1)} C_i\left|q^{(i-1)}\right\rangle\left|\psi_p^{(i-1)}\right\rangle \tag{3.95}$$

$$= C_i\left|\psi_{\text{good}}^{(i-1)}\right\rangle \tag{3.96}$$

$$= \left|\psi_{\text{good}}^{(i)}\right\rangle \tag{3.97}$$

where the first and last steps follow from parts 1 and 2 of Lemma 47, respectively; the second follows from Definition 40; the third follows from Lemma 46; and the fourth follows from the induction hypothesis.

2. $C_i = \wedge(O_c)$ for some $c \in \mathcal{C}$. Then

$$L\left|\phi^{(i)}_{\text{good}}\right\rangle = L\tilde{C}_i\left|\phi^{(i-1)}_{\text{great}}\right\rangle \tag{3.98}$$

$$= \sum_{p\in\mathcal{P}^{(i-1)}_{\text{great}_{\mathcal{A}}}} \sum_q \alpha^{(i-1)}_{p,q} L\tilde{C}_i\left|q^{(i-1)}\right\rangle\left|\phi^{(i-1)}_p\right\rangle \tag{3.99}$$

$$= \sum_{p\in\mathcal{P}^{(i-1)}_{\text{great}_{\mathcal{A}}}} \sum_q \alpha^{(i-1)}_{p,q} C_i\left|q^{(i-1)}\right\rangle\left|\psi^{(i-1)}_p\right\rangle \tag{3.100}$$

$$= C_i\left|\psi^{(i-1)}_{\text{great}}\right\rangle \tag{3.101}$$

$$= \left|\psi^{(i)}_{\text{good}}\right\rangle \tag{3.102}$$

where the first step follows from part 1 of Lemma 52, the second and fourth follow from Definition 48, the third follows from Lemma 46, and the last follows from the induction hypothesis and part 2 of Lemma 52. □

Finally, we can now show the following relationship between the norms of $\left|\phi^{(i)}_{\text{good}}\right\rangle$ and $\left|\psi^{(i)}_{\text{good}}\right\rangle$, which will be very useful in bounding the probability of success of Algorithm $\mathcal{A}$ in Section 3.6.

Lemma 54. *Let $i \in [p(n)] \cup \{0\}$. Then, $\left\|\left|\phi^{(i)}_{\text{good}}\right\rangle\right\| = \left\|\left|\psi^{(i)}_{\text{good}}\right\rangle\right\|$.*

Proof. It is clear, by Definition 40, that for $p, p' \in \mathcal{P}^{(i)}_{\text{good}_{\mathcal{A}}}$, if $p \neq p'$, then $\left|\phi^{(i)}_p\right\rangle \neq \left|\phi^{(i)}_{p'}\right\rangle$. Since $\left|\phi^{(i)}_p\right\rangle$ and $\left|\phi^{(i)}_{p'}\right\rangle$ are computational basis states, this means that $\left\langle\phi^{(i)}_p\middle|\phi^{(i)}_{p'}\right\rangle = 0$ whenever $p \neq p'$. Notice that $\left|\psi^{(i)}_p\right\rangle$ and $\left|\psi^{(i)}_{p'}\right\rangle$ are in the support of $\left|\psi^{(i)}_{\text{good}}\right\rangle$ by Lemma 53. Recall from the discussion in Section 3.4 that any state in the support of $\left|\psi^{(i)}_{\text{good}}\right\rangle$ is rooted. Thus, the contrapositive of Lemma 44 implies that $\left|\psi^{(i)}_p\right\rangle \neq \left|\psi^{(i)}_{p'}\right\rangle$, which essentially means that $\left\langle\psi^{(i)}_p\middle|\psi^{(i)}_{p'}\right\rangle = 0$ since $\left|\psi^{(i)}_p\right\rangle$ and $\left|\psi^{(i)}_{p'}\right\rangle$ are computational basis states. Combining these observations

with Lemma 53, we get

$$\left\| \left| \psi^{(i)}_{\text{good}} \right\rangle \right\| = \left\| L \left| \phi^{(i)}_{\text{good}} \right\rangle \right\| \tag{3.103}$$

$$= \left\| \sum_{p \in \mathcal{P}^{(i)}_{\text{good}_\mathcal{A}}} \sum_q \alpha^{(i)}_{p,q} \left| q^{(i)} \right\rangle \left| \psi^{(i)}_p \right\rangle \right\| \tag{3.104}$$

$$= \sum_{p \in \mathcal{P}^{(i)}_{\text{good}_\mathcal{A}}} \sum_q \left| \alpha^{(i)}_{p,q} \right|^2 \tag{3.105}$$

$$= \left\| \sum_{p \in \mathcal{P}^{(i)}_{\text{good}_\mathcal{A}}} \sum_q \alpha^{(i)}_{p,q} \left| q^{(i)} \right\rangle \left| \phi^{(i)}_p \right\rangle \right\| \tag{3.106}$$

$$= \left\| \left| \phi^{(i)}_{\text{good}} \right\rangle \right\| \tag{3.107}$$

as claimed. □

3.6 The state is mostly good

By the end of this section, we will conclude that it is hard for any rooted genuine quantum algorithm to find the EXIT (and hence, an ENTRANCE–EXIT path). We will achieve this goal by bounding the mass of the quantum state $|\psi_\mathcal{A}\rangle$ associated with any arbitrarily chosen rooted genuine quantum algorithm $\mathcal{A}$ that lies in the ψ-BAD subspace. We will proceed by first using the result of Chapter 4 to bound the mass of the quantum state $\left| \phi^{(i)}_\mathcal{A} \right\rangle$ that lies in the ϕ-BAD subspace.

Lemma 55. *Let $i \in [p(n)] \cup \{0\}$. Then $\left\| \Pi^\phi_{\text{bad}} \left| \phi^{(i)}_\mathcal{A} \right\rangle \right\|^2 \leq 4p(n)^4 \cdot 2^{-n/3}$.*

Proof. By Definition 36, each computational basis state in $\Pi^\phi_{\text{bad}} \left| \phi^{(i)}_\mathcal{A} \right\rangle$ is a ϕ-bad state. Therefore, the ith step of the classical Algorithm 3 outputs a computational basis state $\left| \psi^{(i)} \right\rangle$ corresponding to a subgraph of $\mathcal{G}$ that contains the EXIT or is at most one edge away from containing a cycle using at most $i \leq p(n)$ queries with probability $\left\| \Pi^\phi_{\text{bad}} \left| \phi^{(i)}_\mathcal{A} \right\rangle \right\|^2$. In the case $\left| \psi^{(i)} \right\rangle$ does not contain

a cycle, we can run a depth-first search of length 1 on the subgraph corresponding to $\left|\psi^{(i)}\right\rangle$ using at most i additional queries. Hence, we have found the EXIT or a cycle using at most $2i \leq 2p(n)$ classical queries with probability $\left\|\Pi^{\phi}_{\text{bad}}\left|\phi^{(i)}_{\mathcal{A}}\right\rangle\right\|^2$. Noting that Algorithm 3 has the form of the classical query algorithms considered by Theorem 79, we see that $\left\|\Pi^{\phi}_{\text{bad}}\left|\phi^{(i)}_{\mathcal{A}}\right\rangle\right\|^2 \leq 4p(n)^4 \cdot 2^{-n/3}$. □

From the result of Lemma 55, one might intuitively conjecture that the size of the portion of the state $\left|\phi^{(i)}_{\mathcal{A}}\right\rangle$ after i steps that encountered the EXIT or an almost cycle at some point in its history is small. Indeed, this is the case as we show in the Lemma that follows where we make non-trivial use of our definitions from Section 3.4.

Lemma 56. *For all $i \in [p(n)] \cup \{0\}$, $\left\|\left|\phi^{(i)}_{\text{allbad}}\right\rangle\right\| \leq 2ip(n)^2 \cdot 2^{-n/6}$.*

Proof. We prove the lemma by induction on i. The base case ($i = 0$) is easy to observe as $\left\|\left|\phi^{(0)}_{\text{allbad}}\right\rangle\right\| = 0$. Now, pick any $i \in [p(n)]$ and suppose that the lemma is true for $i-1$. That is, $\left\|\left|\phi^{(i-1)}_{\text{allbad}}\right\rangle\right\| \leq 2(i-1)p(n)^2 \cdot 2^{-n/6}$. Then, by Definitions 36 and 37 and part 9 of Lemma 38,

$$\left|\phi^{(i)}_{\text{allbad}}\right\rangle = \left|\phi^{(i)}_{\text{bad}}\right\rangle + C_i\left|\phi^{(i-1)}_{\text{allbad}}\right\rangle \tag{3.108}$$

$$= \left|\phi^{(i)}_{\text{bad}}\right\rangle + \Pi^{\phi}_{\text{bad}}C_i\left|\phi^{(i-1)}_{\text{allbad}}\right\rangle + \Pi^{\phi}_{\text{good}}C_i\left|\phi^{(i-1)}_{\text{allbad}}\right\rangle \tag{3.109}$$

$$= \Pi^{\phi}_{\text{bad}}\left|\phi^{(i)}_{\mathcal{A}}\right\rangle + \Pi^{\phi}_{\text{good}}C_i\left|\phi^{(i-1)}_{\text{allbad}}\right\rangle, \tag{3.110}$$

so we have

$$\left\| \left| \phi^{(i)}_{\text{allbad}} \right\rangle \right\| = \left\| \Pi^{\phi}_{\text{bad}} \left| \phi^{(i)}_{\mathcal{A}} \right\rangle + \Pi^{\phi}_{\text{good}} C_i \left| \phi^{(i-1)}_{\text{allbad}} \right\rangle \right\| \tag{3.111}$$

$$\leq \left\| \Pi^{\phi}_{\text{bad}} \left| \phi^{(i)}_{\mathcal{A}} \right\rangle \right\| + \left\| \Pi^{\phi}_{\text{good}} C_i \left| \phi^{(i-1)}_{\text{allbad}} \right\rangle \right\| \tag{3.112}$$

$$\leq \left\| \Pi^{\phi}_{\text{bad}} \left| \phi^{(i)}_{\mathcal{A}} \right\rangle \right\| + \left\| \left| \phi^{(i-1)}_{\text{allbad}} \right\rangle \right\| \tag{3.113}$$

$$= \frac{2ip(n)^2}{2^{n/6}} \tag{3.114}$$

where the second step follows by the triangle inequality, the third by the fact that applying a unitary C_i and the projector Π^{ϕ}_{good} cannot increase the norm of any vector, and the fourth by Lemma 55 and the induction hypothesis. □

The bound on the size of the portion of the state $\left| \phi^{(i)}_{\mathcal{A}} \right\rangle$ after i steps that never encountered the EXIT or an almost cycle directly follows from Lemma 56 as shown by the following corollary.

Corollary 57. *Let $i \in [p(n)]$. Then $\left\| \left| \phi^{(i)}_{\text{good}} \right\rangle \right\| \geq 1 - 2ip(n)^2 \cdot 2^{-n/6}$.*

Proof. Observe that

$$\left\| \left| \phi^{(i)}_{\text{good}} \right\rangle \right\| = \left\| \left| \phi^{(i)}_{\mathcal{A}} \right\rangle - \left| \phi^{(i)}_{\text{allbad}} \right\rangle \right\| \geq \left\| \left| \phi^{(i)}_{\mathcal{A}} \right\rangle \right\| - \left\| \left| \phi^{(i)}_{\text{allbad}} \right\rangle \right\| \geq 1 - \frac{2ip(n)^2}{2^{n/6}} \tag{3.115}$$

where the equality follows by Definition 37, the first inequality is an application of the triangle inequality, and the second inequality follows by Lemma 56 and the fact that $\left| \phi^{(i)}_{\mathcal{A}} \right\rangle$ is a quantum state. □

In the next Lemma, we bound the mass on the size of the portion of the state $\left| \psi^{(i)}_{\mathcal{A}} \right\rangle$ after i steps that encountered the EXIT or an almost cycle at some point in its history. This is a crucial Lemma for our result in this section where we invoke Lemma 54 to deduce a statement about

the quantum state of the genuine algorithm $\mathcal{A}$ using the statements that we know about the state of our classical simulation of $\mathcal{A}$.

Lemma 58. *Let $i \in [p(n)]$. Then, $\left\| \left| \psi^{(i)}_{\text{allbad}} \right\rangle \right\|^2 \leq 4i^2 p(n)^2 \cdot 2^{-n/6}$.*

Proof. Observe that

$$\left\| \left| \psi^{(i)}_{\text{allbad}} \right\rangle \right\| = \left\| \sum_{j \in [i]} C_{j,i} \left| \psi^{(j)}_{\text{bad}} \right\rangle \right\| \tag{3.116}$$

$$\leq \sum_{j \in [i]} \left\| C_{j,i} \left| \psi^{(j)}_{\text{bad}} \right\rangle \right\| \tag{3.117}$$

$$= \sum_{j \in [i]} \left\| \left| \psi^{(j)}_{\text{bad}} \right\rangle \right\| \tag{3.118}$$

where the first step follows by Definition 37, the second by triangle inequality, and the third by the fact that the unitary $C_{j,i}$ preserves norms.

Thus, we have

$$\left\| \left| \psi^{(i)}_{\text{allbad}} \right\rangle \right\|^2 \leq \left(\sum_{j \in [i]} \left\| \left| \psi^{(j)}_{\text{bad}} \right\rangle \right\| \right)^2 \tag{3.119}$$

$$\leq \sum_{j \in [i]} i \left\| \left| \psi^{(j)}_{\text{bad}} \right\rangle \right\|^2 \tag{3.120}$$

$$= i \left(1 - \left\| \left| \psi^{(i)}_{\text{good}} \right\rangle \right\|^2 \right) \tag{3.121}$$

$$= i \left(1 - \left\| \left| \phi^{(i)}_{\text{good}} \right\rangle \right\|^2 \right) \tag{3.122}$$

$$\leq i \left(1 - \left(1 - \frac{2ip(n)^2}{2^{n/6}} \right)^2 \right) \tag{3.123}$$

$$\leq \frac{4i^2 p(n)^2}{2^{n/6}} \tag{3.124}$$

where the two equalities follow by Lemmas 39 and 54, respectively, and the next-to-last inequality follows by Corollary 57. □

We are now ready to describe and state our main theorem, which formally proves the hardness of finding an ENTRANCE–EXIT path for genuine, rooted quantum query algorithms.

Theorem 59. *Any genuine, rooted quantum query algorithm for the path-finding problem cannot find a path from* ENTRANCE *to* EXIT *with more than exponentially small probability.*

Proof. Since we let $\mathcal{A}$ be any arbitrary genuine, rooted quantum algorithm, it is sufficient to show that $\mathcal{A}$ cannot find a path from ENTRANCE to EXIT with more than exponentially small probability. Note that, by Definition 35, any computational basis state $|\psi\rangle$ that corresponds to a subgraph that stores an ENTRANCE to EXIT path must be ψ-bad. That is, such a $|\psi\rangle$ must be in the support of $\Pi^{\psi}_{\text{bad}}|\psi_{\mathcal{A}}\rangle$ from Definition 36. Recall from Definition 10 that the genuine algorithm $\mathcal{A}$ measures the state $|\psi_{\mathcal{A}}\rangle$ and output the resulting set of vertices. Thus, the probability that the genuine, rooted quantum query algorithm $\mathcal{A}$ finds an ENTRANCE to EXIT path is at most

$$\begin{aligned}
\left\|\Pi^{\psi}_{\text{bad}}|\psi_{\mathcal{A}}\rangle\right\|^2 &= \left\|\Pi^{\psi}_{\text{bad}}\left|\psi^{(p(n)}_{\mathcal{A}}\right\rangle\right\|^2 && (3.125)\\
&= \left\|\Pi^{\psi}_{\text{bad}}\left|\psi^{(p(n)}_{\text{good}}\right\rangle + \Pi^{\psi}_{\text{bad}}\left|\psi^{(p(n)}_{\text{allbad}}\right\rangle\right\|^2 && (3.126)\\
&= \left\|\Pi^{\psi}_{\text{bad}}\left|\psi^{(p(n)}_{\text{allbad}}\right\rangle\right\|^2 && (3.127)\\
&\leq \left\|\Pi^{\psi}_{\text{good}}\left|\psi^{(p(n)}_{\text{allbad}}\right\rangle\right\|^2 + \left\|\Pi^{\psi}_{\text{bad}}\left|\psi^{(p(n)}_{\text{allbad}}\right\rangle\right\|^2 && (3.128)\\
&= \left\|\Pi^{\psi}_{\text{good}}\left|\psi^{(p(n)}_{\text{allbad}}\right\rangle + \Pi^{\psi}_{\text{bad}}\left|\psi^{(p(n)}_{\text{allbad}}\right\rangle\right\|^2 && (3.129)\\
&= \left\|\left|\psi^{(p(n)}_{\text{allbad}}\right\rangle\right\|^2 && (3.130)\\
&\leq \frac{4p(n)^4}{2^{n/6}} && (3.131)
\end{aligned}$$

where we used Definition 37 in the first step, part 6 of Lemma 38 in the second, part 2 of Lemma 38 in the third, Definition 36 in the fifth and the sixth steps, and Lemma 58 in the last.

□

Chapter 4: Classical hardness for 3-color oracle

In this chapter, we analyze the classical query complexity of finding the EXIT or a cycle in a randomly chosen 3-colored Welded Tree Graph of size n. More precisely, we show that the probability of finding the EXIT or a cycle for a natural class of classical algorithms is exponentially small (Theorem 79) even for a Welded Tree Graph whose vertices are permuted according to the distribution D_n specified in Definition 66 below. Informally, D_n gives rise to the uniform distribution on Welded Tree Graphs over the set that is constructed by fixing a 3-colored Welded Tree Graph $\mathcal{G}$ and randomizing the vertices of the WELD cycle (defined in Definition 64) making sure that the resulting graphs are valid 3-colored Welded Tree Graphs.

The key ingredient of our analysis is Lemma 74 (see also Corollary 75), which informally says that for a Welded Tree Graph sampled according to the aforementioned distribution, it is exponentially unlikely for a certain natural class of classical algorithms (i) to get 'close' to the ENTRANCE or the EXIT starting on any vertex in the WELD cycle without backtracking, or (ii) to encounter two WELD vertices that are connected by multiple 'short' paths. Note that statement (i) implies that it is hard for any such classical algorithm to find the EXIT whereas statement (ii) has a similar implication for finding a cycle. An astute reader might notice the resemblance of Lemma 74 with Lemma 8 of [1]. Indeed, this Lemma proved that it is hard for any classical algorithm with access to a colorless Welded Tree Oracle to satisfy either statements (i) and (ii) mentioned above. However, the argument of [1] is different than ours in two major ways: our prove is by induction; we use randomness of the WELD cycle to argue the unlikeliness

of statement (i) while they use hardness of guessing multiple coin tosses.

We begin by defining the notation that we will use to refer to color sequences that will specify paths in binary trees (and the Welded Tree).

Definition 60. *Let T be a binary tree of height n, which is edge-colored using the 3 colors in $\mathcal{C}$. Let v be any leaf of T, $j \in [n]$, and u be the ancestor of v in T that is distance j away from v. We define t_v^j to be the length-j sequence of colors from v to u.*

The following Lemma formalizes the observation that the color of edges at any level of a binary tree are almost-uniformly distributed among the possible 3 colors.

Lemma 61. *Let T be a binary tree of height n, which is edge-colored using the 3 colors in $\mathcal{C}$. Let $c_* \in \mathcal{C}$ denote the unique color such that there is no c_*-colored edge incident to the root of T. For each $c \in \mathcal{C}$ and $i \in [n]$, let $\gamma(c, i)$ denote the number of c-colored edges at level i. Then*

$$\gamma(c,i) = \begin{cases} \lfloor 2^i/3 \rfloor & i \text{ is odd and } c = c_* \text{ or } i \text{ is even and } c \neq c_* \\ \lceil 2^i/3 \rceil & \text{otherwise.} \end{cases} \tag{4.1}$$

Proof. We prove this Lemma by induction on i. An edge is at level 1 in T iff it is incident to the root of T. The base case of the Lemma directly follows.

Assume that this Lemma is true for some $i \in [n-1]$. Suppose that i is odd. Note that, for any $c \in \mathcal{C}$, any vertex in column i is connected to a vertex in column $i+1$ with a c-colored edge if and only if it is connected to a vertex in column $i-1$ with a c'-colored edge for $c \neq c'$. Then,

the number of c-colored edges at level $i+1$ are

$$\gamma(c, i+1) = \sum_{\substack{c' \in \mathcal{C} \\ c' \neq c}} \gamma(c', i) \tag{4.2}$$

$$= \begin{cases} 2\lceil 2^i/3 \rceil & c = c_* \\ \lceil 2^i/3 \rceil + \lfloor 2^i/3 \rfloor & \text{otherwise} \end{cases} \tag{4.3}$$

$$= \begin{cases} \lceil 2^{i+1}/3 \rceil & c = c_* \\ \lfloor 2^{i+1}/3 \rfloor & \text{otherwise.} \end{cases} \tag{4.4}$$

The analysis for even i is very similar. □

Lemma 61 directly imply the following corollary, which informally states that the number of paths from a particular level $n-j$ to the leaves of a binary tree are almost-uniformly distributed among all possible color sequences of length j that does not contain an even-length palindrome.

Corollary 62. *Let T be a binary tree of height n, which is edge-colored using the 3 colors in $\mathcal{C}$. Let $j \in [n]$ and fix a length j sequence of colors $t \in \mathcal{C}^j$ that does not contain an even-length palindrome. Then, the number of leaves v of T satisfying $t_v^j = t$ is at most $\lceil 2^{n-j+1}/3 \rceil$.*

Proof. Let $c \in \mathcal{C}$ be the last color appearing in the sequence t. Note that any ancestor of a leaf v of T that is distance j away from v must be in column $n-j$ of T. For each vertex u in column $n-j$ with some edge at level $n-j+1$ incident to u being c-colored, there is exactly one leaf v such that v is a descendant of u and $t_v^j = t$. By Lemma 61, there are at most $\lceil 2^{n-j+1}/3 \rceil$ c-colored edges at level $n-j+1$. Therefore, there are at most $\lceil 2^{n-j+1}/3 \rceil$ c leaves of T satisfying $t_v^j = t$. □

Consider the following consequence of Lemma 61, which is an interesting observation about valid 3-colorings of Welded Tree Graphs even though it is not essential for the argument

of this chapter. However, it implies that Figure 2.1 is not an example with loss of generality.

Lemma 63. *For any valid 3-coloring of $\mathcal{G}$, c_* is the unique color with no edge incident to the* EXIT.

Proof. Without loss of generality, assume that n is odd. Recall that c_* referred to the unique color with no edge incident to the ENTRANCE. By Lemma 61, the number of edges of color c_* in $\mathcal{G}$ at level n are $\lfloor 2^n \rfloor / 3$. This means that the number of non-c_* edges at level n are $2^n - \lfloor 2^n \rfloor / 3 = 2\lceil 2^n \rceil / 3$. Note that there is a bijection between the non-c_* edges at level n and c_* edges at level $n+1$. Therefore, there must be $2\lceil 2^n \rceil / 3$ edges of color c_* at level $n+1$.

Now, suppose that there is a c_* edge incident to the EXIT. Thus, by Lemma 61, the number of edges of color c_* in $\mathcal{G}$ at level $n+2$ must be $\lceil 2^n \rceil / 3$. It follows, by a similar argument as in the above paragraph, that the number of c_* edges at level $n+1$ are $2^n - \lceil 2^n \rceil / 3 = \lceil 2^n \rceil / 3 + \lfloor 2^n \rfloor / 3$. But this contradicts an above assertion. Hence, c_* is the unique color with no edge incident to the EXIT. □

Consider the following induced subgraphs of $\mathcal{G}$.

Definition 64. *Define T_L, T_R, and* WELD *to be the induced subgraphs of $\mathcal{G}$ on vertices in columns $\{0, \ldots, n\}$, columns $\{n+1, \ldots, 2n+1\}$, and columns $\{n, n+1\}$ of $\mathcal{G}$, respectively.*

Informally, T_L and T_R are induced subgraphs of $\mathcal{G}$ on vertices in the left and right binary trees of $\mathcal{G}$, respectively, while WELD is the induced subgraph of $\mathcal{G}$ on the leaves of the left and right binary trees of $\mathcal{G}$. Note that T_L and T_R are height-n subtrees of $\mathcal{G}$ rooted at ENTRANCE and EXIT, respectively. Furthermore, the vertices of T_L and T_R provide a bipartition of the vertices of $\mathcal{G}$, and the edges of T_L, T_R, and WELD provide a tripartition of the edges of $\mathcal{G}$. Now, we categorize the vertices of WELD depending on whether it is a vertex of T_L or T_R and the color of edge that it is joined with to a non-WELD vertex.

Definition 65. *For any leaf v of the tree T_L and $c \in \mathcal{C}$, if the color of the edge connecting v with T_L in $\mathcal{G}$ is c, then we say that v is a c-left vertex. Similarly, we define the notion of a c-right vertex for each $c \in \mathcal{C}$.*

As an example, the vertices colored lavender and plum in Figure 4.1(a) are red-left vertices. Next, we define permutations that map valid 3-colored Welded Tree Graphs to valid 3-colored Welded Tree Graphs as we show in Lemma 67. Note that Definition 65 partitions the vertices of WELD into 6 parts. The following definition is crucial for the description of the distribution of Welded Tree Graphs that is classically 'hard'.

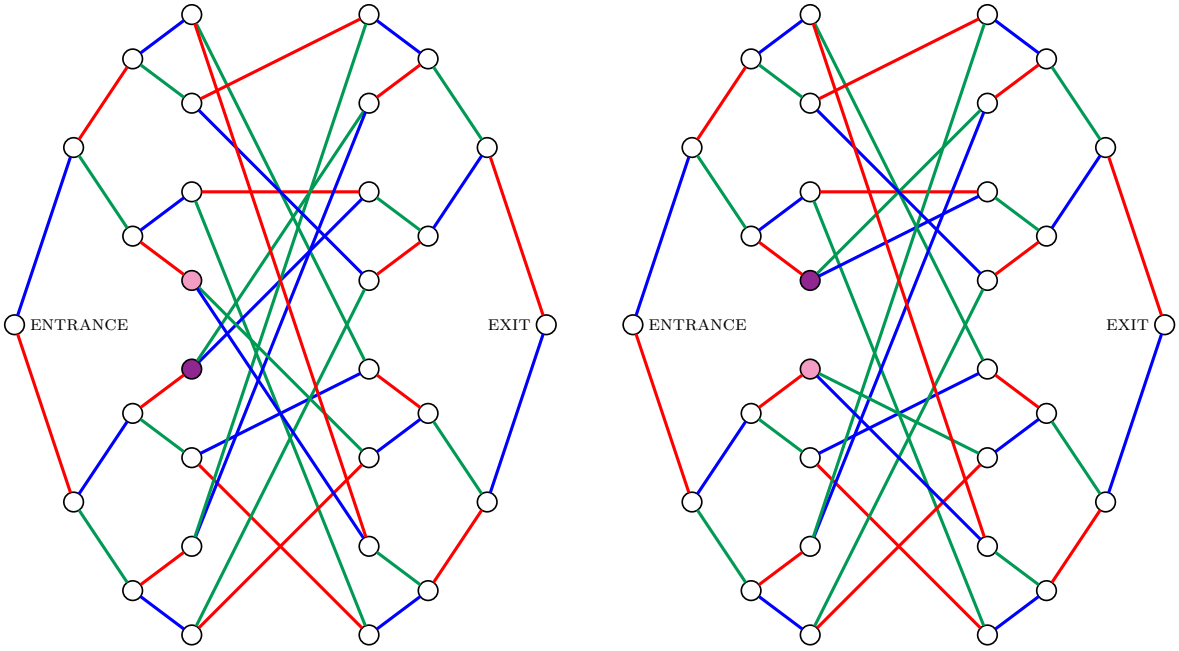
Definition 66 (Color-preserving permutations)**.** *For any permutation σ of the vertices of $\mathcal{G}$, let $\mathcal{G}^\sigma$ denote the graph obtained by applying σ to the vertices of $\mathcal{G}$: there is an edge of color $c \in \mathcal{C}$ in $\mathcal{G}$ joining vertices u and v if and only if there is an edge of color c in $\mathcal{G}^\sigma$ joining vertices $\sigma(u)$ and $\sigma(v)$. A permutation σ of the vertices of $\mathcal{G}$ is a* color-preserving permutation *if*

1. *for any vertex v of $\mathcal{G}$ that is not a vertex of* WELD*, $\sigma(v) = v$, and*
2. *for any vertex v of* WELD*, $\sigma(v)$ is a vertex of* WELD*, and for any $c \in \mathcal{C}$, if v is c-left (respectively c-right), then $\sigma(v)$ is c-left (respectively c-right).*

For any color-preserving permutation σ, let T_L^σ (respectively WELD^σ*, T_R^σ) denote the graph obtained by applying σ restricted to the vertices of T_L (respectively* WELD*, T_R) to the vertices of T_L (respectively* WELD*, T_R). Define D_n to be the uniform distribution over all color-preserving permutations σ.*

Figure 4.1 illustrates an example of a color-preserving permutation. Note that the non-WELD vertices remain invariant under σ. Therefore, WELD^σ is the induced subgraph of $\mathcal{G}^\sigma$ on vertices in columns $\{n, n+1\}$ of $\mathcal{G}^\sigma$.

For each $c \in \mathcal{C}$, by Lemma 61, we have at least $\lfloor 2^n/3 \rfloor$ c-left and at least $\lfloor 2^n/3 \rfloor$ c-right vertices in $\mathcal{G}$. Hence, there are at least $(\lfloor 2^n/3 \rfloor !)^6$ color-preserving permutations.



(a) $\mathcal{G}$ (b) $\mathcal{G}^\sigma$

Figure 4.1: Example of a color-preserving permutation σ for the graph $\mathcal{G}$ in Figure 2.1. The permutation σ is the identity permutation except that it maps the vertex colored lavender to the vertex colored plum. Note that the resulting graph G^σ is a valid 3-colored Welded Tree Graph.

We now verify that the graphs obtained by applying color-preserving permutations on $\mathcal{G}$ are valid 3-colored Welded Tree Graphs.

Lemma 67. *Let σ be a color-preserving permutation. Then $\mathcal{G}^\sigma$ is a valid 3-colored Welded Tree Graph.*

Proof. We first argue that $\mathcal{G}^\sigma$ is a valid Welded Tree Graph. Recall that the vertices of $\mathcal{G}$ that are not in the WELD remain invariant under σ. Thus, it remains to show that WELD^σ is a cycle alternating between the vertices in the columns n and $n+1$ of $\mathcal{G}^\sigma$. Since $\mathcal{G}$ is a Welded Tree Graph, WELD is a cycle on vertices denoted by $v_1, \ldots, v_{2^{n+1}}$ such that v_i is joined to $v_{i+1 \bmod 2^{n+1}}$ and $v_{i-1 \bmod 2^{n+1}}$ for each $i \in [2^{n+1}]$. Therefore, for each $i \in [2^{n+1}]$, $\sigma(v_i)$ is joined to $\sigma(v_{i+1 \bmod 2^{n+1}})$ and $\sigma(v_{i-1 \bmod 2^{n+1}})$. It follows that WELD^σ is a cycle on vertices $\sigma(v_1), \ldots, \sigma(v_{2^{n+1}})$.

Now, we claim that $\mathcal{G}^\sigma$ admits a valid 3-coloring. Let v be any vertex of WELD. Without loss of generality, assume that v is red-left. Then, as $\mathcal{G}$ is a valid 3-colored graph and v has degree

3, v is joined to a vertex v_r of T_L with a red edge and two vertices v_g and v_b of WELD with a green and a blue edge respectively. Since σ is color-preserving, $\sigma(v)$ is joined to v_r with a red edge and to $\sigma(v_g)$ and $\sigma(v_b)$ of WELD^σ with a green and a blue edge respectively. On the other hand, non-WELD vertices of $\mathcal{G}$ are unchanged by σ. Our desired claim follows. □

Recall that Definition 3 defined the classical oracle function η_c^σ for each $c \in \mathcal{C}$ associated with $\mathcal{G}^\sigma$ for the identity permutation σ. The following definition generalizes it by specifying the classical oracle function η_c^σ for each $c \in \mathcal{C}$ associated with $\mathcal{G}^\sigma$ for any color-preserving permutation σ.

Definition 68. *Let $\mathcal{V}_\mathcal{G}$, I_c, and N_c for each $c \in \mathcal{C}$, NOEDGE, and INVALID be defined as in Definitions 1 and 3. For any color-preserving permutation σ, let $N_c^\sigma(v) := \sigma(N_c(\sigma^{-1}(v))$. Then, let*

$$\eta_c^\sigma(v) := \begin{cases} N_c^\sigma(v) & v \in \mathcal{V}_\mathcal{G} \text{ and } I_c(v) = 1 \\ \text{NOEDGE} & v \in \mathcal{V}_\mathcal{G} \text{ and } I_c(v) = 0 \\ \text{INVALID} & v \notin \mathcal{V}_\mathcal{G}, \end{cases} \tag{4.5}$$

Let $\eta^\sigma := \{\eta_c^\sigma : c \in \mathcal{C}\}$ be the oracle corresponding to the color-preserving permutation σ.

For any color-preserving permutation σ, as ENTRANCE and EXIT remain invariant under σ, for any $v \in \mathcal{V}_\mathcal{G}$ and $c \in \mathcal{C}$ with $I_c(v) = 1$, we have $N_c^\sigma(v) \in \mathcal{V}_\mathcal{G}$. Next, we consider $2^{2n/3}$ disjoint subtress of T_L (respectively T_R), each of which contains (as leaves) $2^{n/3}$ consecutive leaves of T_L (respectively T_R).

Definition 69. *Let σ be any color-preserving permutation. Fix an ordering of the vertices of T_L^σ (respectively T_R^σ) in column $2n/3$. For any $i \in [2^{2n/3}]$, let $T_{L_i}^\sigma$ (respectively $T_{R_i}^\sigma$) denote the subtree of T_L^σ (respectively T_R^σ) that is a binary tree of height $n/3$ rooted at the ith vertex in column $2n/3$ of T_L^σ (respectively T_R^σ). Moreover, let $\mathbb{T}^\sigma := \{T_{L_i}^\sigma, T_{R_i}^\sigma : i \in [2^{2n/3}]\}$.*

Observe that all the non-leaf vertices of T_L^σ and T_R^σ are invariant under the choice of σ. We now define the notion of path-embedding in $\mathcal{G}^\sigma$ for a sequence of colors t, which informally refers to the path resulting from beginning at the ENTRANCE and following the edge colors given by t in order.

Definition 70 (Path-embedding)**.** *Let σ be any color-preserving permutation. Let $\ell \in [p(n)]$ and $t \in \mathcal{C}^\ell$. That is, $t = (c_1, \ldots, c_\ell)$ for some $c_1, \ldots, c_\ell \in \mathcal{C}$. Then, define the* path-embedding *of t under the oracle η^σ, denoted by $\eta^\sigma(t)$, to be a length-ℓ tuple of vertex labels as follows. The jth element of $\eta^\sigma(t)$ is*

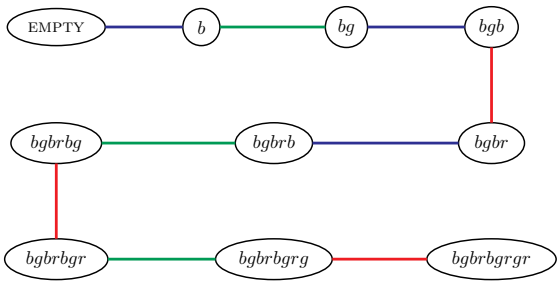
$$\eta^\sigma(t)_j = \begin{cases} \eta^\sigma_{c_1}(\text{ENTRANCE}) & j = 1 \\ \eta^\sigma_{c_j}(\eta^\sigma(t)_{j-1}) & \textit{otherwise.} \end{cases} \tag{4.6}$$

We say that the path-embedding $\eta^\sigma(t)$ encounters a vertex *v if $\eta^\sigma(t)_j = v$ for some $j \in [\ell]$ and $\eta^\sigma(t)$* encounters an edge *joining vertices v and u if $\eta^\sigma(t)_j = v$ and $\eta^\sigma(t)_{j+1} = u$ (or the other way around) for some $j \in [\ell - 1]$. Moreover, $\eta^\sigma(t)$* encounters a cycle *in $\mathcal{G}^\sigma$ if it encounters a sequence of vertices and edges that forms a cycle in $\mathcal{G}^\sigma$ and* encounters a tree from $\mathbb{T}^\sigma$ *if it encounters a leaf of this tree.*

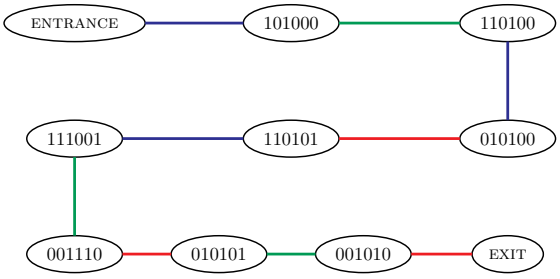
Figure 4.2 demonstrates an example of a path in $\mathcal{T}$ and the corresponding path-embedding in $\mathcal{G}$. One can interpret $\eta^\sigma(t)$ as a tuple of vertex labels of length ℓ obtained by starting at the ENTRANCE, and following the edge colors listed in t. We will only restrict our attention to the color sequences that does not contain even-length palindromes. For such a sequence t, the path-embedding $\eta^\sigma(t)$ encounters a cycle exactly when it encounters a vertex twice.

Now, we will describe the notation that we will use for each time a certain path-embedding crosses the WELD to refer to the tree from $\mathbb{T}^\sigma$ that it goes to and the WELD edge that it goes through.

Definition 71. *Let σ be any color-preserving permutation and t be any sequence of colors that does not*



(a) The path in $\mathcal{T}$ given by t



(b) The path-embedding $\eta^\sigma(t)$ of t in $\mathcal{G}$

Figure 4.2: Example of a path-embedding for the graph $\mathcal{G}$ in Figure 2.1 and the identity permutation σ.

contain even-length palindromes. We use $T^\sigma_{t,i} \in \mathbb{T}^\sigma$ *to denote the* i*th subtree and* $e^\sigma_{t,i}$ *to denote the* i*th edge of* WELD *encountered by the path-embedding* $\eta^\sigma(t)$.[1] *We refer to the event of the path-embedding* $\eta^\sigma(t)$ *encountering the* i*th edge of* WELD *as the* i*th* WELD*-crossing. Furthermore, let* $\ell^\sigma(t)$ *denote the number of subtrees encountered by the path-embedding* $\eta^\sigma(t)$.

Note that the number of edges of WELD encountered by the path-embedding $\eta^\sigma(t)$ is $\ell^\sigma(t) - 1$. For each $i \in [\ell^\sigma(t) - 1]$, the edge $e^\sigma_{t,i}$ joins a vertex in $T^\sigma_{t,i}$ to a vertex in $T^\sigma_{t,i+1}$. Next, we define $\ell^\sigma(t)$ prefixes of t that are relevant for our analysis. Intuitively, for $i \in [\ell^\sigma(t) - 1]$, the sequence of colors $\mathsf{pre}^\sigma_i(t)$ refers to the prefix of t such that if we begin from the ENTRANCE and follow the edge colors given by $\mathsf{pre}^\sigma_i(t)$, we will end up on the vertex reached by the ith edge of WELD encountered by $\eta^\sigma(t)$.

Definition 72. *Let* σ *be any color-preserving permutation and* t *be any sequence of colors that does not contain even-length palindromes. Let* $\mathsf{pre}^\sigma_{\ell^\sigma(t)}(t) = t$. *For each* $i \in [\ell^\sigma(t) - 1]$, *let* $\mathsf{pre}^\sigma_i(t)$ *denote the smallest prefix of* t *such that the path-embedding* $\eta^\sigma(\mathsf{pre}^\sigma_i(t))$ *encounters the* i*th* WELD*-crossing.*

Notice that $\mathsf{pre}^\sigma_i(t)$ is a sequence that begins with the color of an edge incident to ENTRANCE and ends with the color of the edge $e^\sigma_{t,i}$. The following definition formalizes statements (i) and (ii) that we intuitively alluded to in the beginning of this chapter.

[1] It is possible for $\eta^\sigma(t)$ to encounter NOEDGE or INVALID. However, once that happens, $\eta^\sigma(t)$ cannot further encounter any subtree from $\mathbb{T}^\sigma$.

Definition 73. *Let σ be any color-preserving permutation and t be any sequence of colors that does not contain even-length palindromes. We say that t has* small displacement *if after the 1^{st}* WELD*-crossing, the path-embedding $\eta^{\sigma}(t)$ does not encounter any vertex that is distance at least $n/3$ away from the closest vertex of* WELD*, and t is* non-colliding *if there is an edge e joining some leaf of $T^{\sigma}_{t,i}$ with some leaf of $T^{\sigma}_{t,j}$ for any $i, j \in [\ell^{\sigma}(t)]$, then e must be $e^{\sigma}_{t,k}$ for some $k \in [\ell^{\sigma}(t) - 1]$. We say that t is* desirable *if t has small displacement and t is non-colliding. We also say that t has* large displacement *if it does not has* small displacement*, t is* colliding *if it is not* non-colliding*, and t is* undesirable *if it is not* desirable*.*

It is easy to observe that for any any sequence t of colors that does not contain even-length palindromes, beginning from the ENTRANCE and following the sequence of colors given by t will not result in reaching the EXIT if t has small displacement, and in going through a cycle if t is non-colliding. The following Lemma is pivotal to our argument in this chapter where we essentially prove that any prefix of any fixed sequence of colors t is unlikely to have large displacement or be colliding (as defined in Definition 73).

Lemma 74. *Let $\ell \in [p(n)]$ and $t \in \mathcal{C}^{\ell}$ such that t does not contain even-length palindromes. Let the permutation σ be chosen according to the distribution D_n. Then, for all $i \in [\ell^{\sigma}(t)]$, $\mathsf{pre}_i(t)$ is desirable with probability at least $1 - 4i^2 2^{-n/3}$ over the choice of σ.*

Proof. We prove the desired statement by induction on i. For the base case, note that $\mathsf{pre}^{\sigma}_1(t)$ does not encounter any edges in any tree in $\mathbb{T}^{\sigma}$ other than the one it first reaches. Therefore, $\mathsf{pre}^{\sigma}_1(t)$ is desirable with certainty.

As the induction hypothesis, assume that $\mathsf{pre}^{\sigma}_i(t)$ is desirable with probability at least $1 - 2i^2 2^{-n/3}$ for some $i \in [\ell^{\sigma}(t) - 1]$. Actually, we assume that $\mathsf{pre}^{\sigma}_i(t)$ and $\eta^{\sigma}(\mathsf{pre}^{\sigma}_i(t))$ are known (and fixed), and $\mathsf{pre}^{\sigma}_i(t)$ has small displacement and is non-colliding with certainty. We union bound the probability of that not happening later.

By our assumption, $\mathsf{pre}^{\sigma}_i(t)$ does not depend on σ so we denote it by $\mathsf{pre}_i(t)$ for simplicity.

By the definition of being non-colliding in Definition 73, we can note that $T^{\sigma}_{t,i+1} \neq T^{\sigma}_{t,j}$ for any $j \in [i]$; otherwise, since $T^{\sigma}_{t,i+1}$ and $T^{\sigma}_{t,i}$ are connected via the edge $e^{\sigma}_{t,i}$, $\mathsf{pre}_i(t)$ would not be non-colliding.

Consider the set $\Delta_i(t) = \{\rho : \eta^{\rho}(\mathsf{pre}^{\rho}_i(t)) = \eta^{\sigma}(\mathsf{pre}_i(t))$ and $T^{\rho}_{t,i+1} \neq T^{\sigma}_{t,j} \forall j \in [i]\}$. We know, from above, that $\sigma \in \Delta_i(t)$. Moreover, any $\rho \in \Delta_i(t)$ is consistent with the induction hypothesis. Since σ is drawn from D_n, it follows that, conditioned on the induction hypothesis, σ is any permutation in $\Delta_i(t)$ with uniform probability.

For any permutation ρ and $j \in [\ell^{\rho}(t)]$, let v^{ρ}_j be the vertex reached by the jth WELD-crossing with respect to the path-embedding $\eta^{\rho}(t)$. Without loss of generality, assume that v^{σ}_{i+1} is a red-right vertex.[2] Let $u \neq v^{\sigma}_{i+1}$ be any red-right leaf of a tree T in $\mathbb{T}^{\sigma}$ such that $T \neq T^{\sigma}_{t,j}$ for all $j \in [i]$. Let ρ be the color-preserving permutation that is the composition with σ of the permutation that maps u to v^{σ}_{i+1} (and vice versa) and is identity otherwise. Since the path-embeddings $\eta^{\sigma}(t)$ and $\eta^{\rho}(t)$ does not encounter vertices v^{σ}_{i+1} and u before the ith WELD-crossing, we have $\eta^{\sigma}(\mathsf{pre}^{\rho}_i(t)) = \eta^{\rho}(\mathsf{pre}_i(t))$. Furthermore, as v^{σ}_{i+1} and u are not leaves of any tree in $\{T^{\sigma}_{t,j} : j \in [i]\}$, we can observe that $T^{\rho}_{t,j} = T^{\sigma}_{t,j}$ for all $j \in [i]$. Therefore, u is a leaf of a tree $T \in \mathbb{T}^{\rho}$ such that $T \neq T^{\sigma}_{t,j}$ for all $j \in [i]$. Hence, $\rho \in \Delta_i(t)$. It follows that for any red-right vertex u that is not a leaf of any tree in $\{T^{\sigma}_{t,j} : j \in [i]\}$, the number of permutations ρ such that $v^{\rho}_{i+1} = u$ are the same.

Let $\mathsf{pre}^{\sigma}_{<i+1}(t)$ denote the sequence resulting from removing the last color from $\mathsf{pre}^{\sigma}_{i+1}(t)$. Note that, since the path-embedding $\eta^{\sigma}(\mathsf{pre}^{\sigma}_{<i+1}(t))$ does not encounter any edge of WELD after the ith WELD-crossing and the non-WELD edges remain invariant under σ, $\mathsf{pre}^{\sigma}_{<i+1}(t)$ does not depend on σ so we write it as $\mathsf{pre}_{<i+1}(t)$. Now, let $\mathsf{sub}_i(t)$ denote the largest suffix of $\mathsf{pre}_{<i+1}(t)$ such that the path-embedding $\eta^{\sigma}(\mathsf{sub}_i(t))$ does not encounter the edge $e^{\sigma}_{t,i}$ (if it exists). This

[2]The same argument follows for any c-right or c-left vertex for any $c \in \mathcal{C}$.

means that $\mathsf{sub}_i(t)$ is a sequence that begins with the color of the edge encountered by the path-embedding $\eta^\sigma(t)$ just after the ith WELD-crossing and ends with the last color of t if $i = \ell^\sigma(t) - 1$ and with the color of the edge encountered by the path-embedding $\eta^\sigma(t)$ just before the $(i+1)$st WELD-crossing otherwise. Let $\ell_i(t)$ denote the length of $\mathsf{sub}_i(t)$.

Observe that the vertex v_i^σ does not depend on σ since we know the path-embedding $\eta^\sigma(\mathsf{pre}_i(t))$. Let $\mathsf{suc}_i^\sigma(t)$ be the sequence of $n/3$ colors beginning from v_i and reaching a vertex in the column $2n/3$ of T_L (if v_i is a leaf in T_L) or T_R (if v_i is a leaf in T_R). Note that, unlike $\mathsf{sub}_i(t)$, $\mathsf{suc}_i^\sigma(t)$ depends on the choice of σ as it is possible for two distinct red-right vertices, for instance, to have distinct color sequences that lead to their respective ancestor in the column $2n/3$ of T_R.

Let $\mathsf{sub}_i(t, n/3)$ denote the prefix of length $n/3$ of $\mathsf{sub}_i(t)$. The sequence $\mathsf{sub}_i(t, n/3)$ does not exist if $|\mathsf{pre}_{i+1}(t)| < |\mathsf{pre}_i(t)| + n/3$. But in that case, we know that $\mathsf{sub}_i(t)$, and hence $\mathsf{pre}_{i+1}(t)$, only encounter vertices that are distance less than $n/3$ away from v_i so $\mathsf{pre}_{i+1}(t)$ has small displacement. Therefore, we assume that $\mathsf{sub}_i(t, n/3)$ exists. Note that, by the induction hypothesis, in order to show that $\mathsf{pre}_i^\sigma(t)$ does has large displacement, we only need to show that the sequence $\mathsf{sub}_i(t)$ beginning from v_i does not encounter any vertex that is distance at least $n/3$ away from v_i. In other words, the probability that $\mathsf{pre}_{i+1}^\sigma(t)$ does not has large displacement is bounded by the probability, over σ, of $\mathsf{suc}_i^\sigma(t) = \mathsf{sub}_i(t, n/3)$, which we compute as follows.

We established above that σ can be any uniformly random permutation in $\Delta_i(t)$. Thus, the probability of $v_{i+1}^\sigma = u$ is the same for all red-right vertices u that are not leaves of any tree in $\{T_{t,j}^\sigma : j \in [i]\}$. It means that the required probability is upper-bounded by the ratio of the number of red-right vertices that satisfy $\mathsf{suc}_i^\sigma(t) = \mathsf{sub}_i(t, n/3)$ and the number of those consistent with the induction hypothesis. By Corollary 62, the number of red-right leaves satisfying $\mathsf{suc}_i^\sigma(t) = \mathsf{sub}_i(t, n/3)$ are at most $2^{2n/3+1}/3 + 1$. On the other hand, by Lemma 61, the total number of red-right vertices that are not leaves of any tree in $\{T_{t,j}^\sigma : j \in [i]\}$ are at least $2^n/3 - 1 - i \cdot 2^{n/3}$ as

any tree in this set has $2^{n/3}$ leaves. Therefore, assuming that $\mathsf{pre}_i^\sigma(t)$ is desirable, the probability $\mathbb{P}_\sigma[E_{i+1}(\text{large displacement})]$ of $\mathsf{pre}_{i+1}^\sigma(t)$ having large displacement is

$$\mathbb{P}_\sigma[E_{i+1}(\text{large displacement})] \leq \mathbb{P}_\sigma[\mathsf{suc}_i^\sigma(t) = \mathsf{sub}_i(t, n/3)] \leq \frac{2^{2n/3+1}/3+1}{2^n/3-1-i\cdot 2^{n/3}} \leq \frac{4}{2^{n/3}} \quad (4.7)$$

where the last inequality follows since there can be at most $p(n)$ WELD-crossings so $i < \ell^\sigma(t) \leq \ell \leq p(n)$.

In order to bound the probability of $\mathsf{pre}_{i+1}^\sigma(t)$ being non-colliding, by the assumption we made for the induction hypothesis, it remains to bound the probability of the tree $T_{t,i+1}^\sigma$ not having an edge with any of the trees in $\{T_{t,j}^\sigma : j \in [i]\}$ other than the edge $e_{t,i}^\sigma$. Pick any leaf u of any of the trees in $\{T_{t,j}^\sigma : j \in [i]\}$. Let w be any neighbor of u that is a vertex of WELD^σ. Without loss of generality, assume that u is green-left and w is red-right. If the edge joining u with w appears in the path-embedding $\eta^\sigma(\mathsf{pre}_{i+1}(t))$, then either this edge is $e_{t,i}^\sigma$ or w is not a leaf of $T_{t,i+1}^\sigma$ as we established above that $T_{t,i+1}^\sigma \neq T_{t,j}^\sigma$ for any $j \in [i]$. Thus, we suppose that it does not appear in $\eta^\sigma(\mathsf{pre}_{i+1}(t))$. Analogous to an argument we made above, the probability of $w = u'$ is the same for all red-right vertices u' that are not leaves of any tree in $\{T_{t,j}^\sigma : j \in [i]\}$. This means that the probability of w being a leaf of $T_{t,i+1}^\sigma$ is bounded by the ratio of the number of red-right vertices in $T_{t,i+1}^\sigma$ and the number of those not leaves of any tree in $\{T_{t,j}^\sigma : j \in [i]\}$. By Lemma 61, the former quantity is at most $2^{n/3}/3+1$ and the latter quantity is at least $2^n/3-1-i\cdot 2^{n/3}$ as all trees in $\{T_{t,j}^\sigma : j \in [i]\}$ have $2^{n/3}$ leaves. Recall that each of the $i\cdot 2^{n/3}$ vertices that are leaves of any tree in $\{T_{t,j}^\sigma : j \in [i]\}$ has 2 neighbours in WELD^σ. Hence, assuming that $\mathsf{pre}_i^\sigma(t)$ is desirable, the probability $\mathbb{P}_\sigma[E_{i+1}(\text{colliding})]$ of $\mathsf{pre}_{i+1}^\sigma(t)$ being colliding is

$$\mathbb{P}_\sigma[E_{i+1}(\text{colliding})] \leq \frac{2i\cdot 2^{n/3}\cdot(2^{n/3}/3+1)}{2^n/3-1-i\cdot 2^{n/3}} \leq \frac{4i}{2^{n/3}} \quad (4.8)$$

by the union bound.

For each $j \in [\ell^\sigma(t)]$, let $\mathbb{P}_\sigma[E_j(\text{desirable})]$ denote the probability of $\mathsf{pre}^\sigma_j(t)$ being desirable. Notice that $\mathsf{pre}^\sigma_{i+1}(t)$ can only be desirable if $\mathsf{pre}^\sigma_i(t)$ is desirable or $\mathsf{pre}^\sigma_{i+1}(t)$ has large displacement or $\mathsf{pre}^\sigma_{i+1}(t)$ is colliding. Hence, we have

$$\begin{aligned}\mathbb{P}_\sigma[E_{i+1}(\text{desirable})] \leq \mathbb{P}_\sigma[E_i(\text{desirable})] + \mathbb{P}_\sigma[E_{i+1}(\text{large displacement})] \\ + \mathbb{P}_\sigma[E_{i+1}(\text{colliding})] \qquad (4.9)\\ \leq \frac{4i^2}{2^{n/3}} + \frac{4}{2^{n/3}} + \frac{4i}{2^{n/3}} \leq \frac{4(i+1)^2}{2^{n/3}} \qquad (4.10)\end{aligned}$$

where we used eqs. (4.7) and (4.8) for the second inequality. The lemma follows. □

The statement of the next Corollary will be sufficient for proving the classical hardness result of Theorem 79 even though it is a weaker statement about a special case of Lemma 74. Concretely, we show that for a fixed sequence of colors and a uniformly random color-preserving permutation σ, it is improbable for the corresponding path-embedding to contain the EXIT or a path that forms a cycle in $\mathcal{G}^\sigma$.

Corollary 75. *Let $\ell \in [p(n)]$ and $t \in \mathcal{C}^\ell$ such that t does not contain even-length palindromes. Let the permutation σ be chosen according to the distribution D_n. Then, the probability that the path-embedding $\eta^\sigma(t)$ encounters the* EXIT *or a cycle in $\mathcal{G}^\sigma$ is at most $4p(n)^2 \cdot 2^{-n/3}$.*

Proof. Note that, by Definition 73, if $\eta^\sigma(t)$ encounters the EXIT, then t has large displacement and if $\eta^\sigma(t)$ encounters a cycle in $\mathcal{G}^\sigma$, then t is colliding. That is, t is undesirable if it encounters the EXIT or a cycle in $\mathcal{G}^\sigma$. Since $\ell^\sigma_t \leq \ell \leq p(n)$, t is undesirable with probability at most $4(\ell^\sigma_t)^2 2^{-n/3} \leq 4p(n)^2 2^{-n/3}$ over the choice of σ by Lemma 74. □

We can extend the result of Corollary 75 about polynomial-size sequences of colors to

polynomial-size subtrees of the address tree $\mathcal{T}$ (see Definition 24). For this purpose, we define the notion of subtree-embedding of subtrees of $\mathcal{T}$. Intuitively, the subtree-embedding of a tree T describes the subgraph of $\mathcal{G}^\sigma$ obtained by querying the oracle η^σ according to the sequences of colors given by the vertex labels of T.

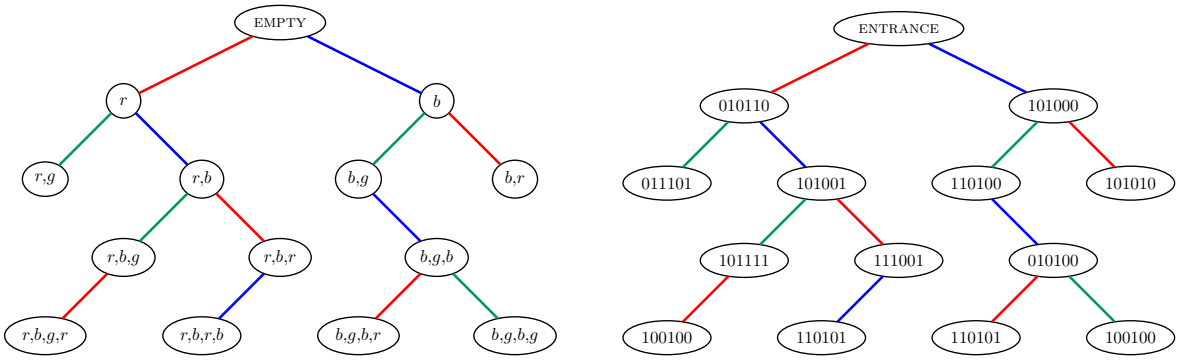
Definition 76 (Subtree-embedding)**.** *Let σ be any color-preserving permutation. Let $\ell \in [p(n)]$ and $t \in \mathcal{C}^\ell$. Let T be a subtree of the address tree $\mathcal{T}$ of size $p(n)$ that contains the vertex labeled* EMPTYADDRESS *but does not the contain vertices having labels in $\mathsf{SpecialAddresses} \setminus \{\text{EMPTYADDRESS}\}$. For any vertex of T labeled by $t \neq$ EMPTYADDRESS, let $c_{|t|}$ denote the last color appearing in the sequence t and $\mathsf{pre}(t)$ denote the color sequence formed by removing the last color from t. Define the subtree-embedding of T under the oracle η^σ, denoted by $\eta^\sigma(T)$, to be a tree isomorphic to T whose vertex labels are in $\mathcal{V}_\mathcal{G}$ and specified as follows. The vertex of $\eta^\sigma(T)$ corresponding to the vertex of T labeled by t is*

$$\eta^\sigma(T)_t := \begin{cases} \text{ENTRANCE} & t = \text{EMPTYADDRESS} \\ \eta^\sigma_{c_{|t|}}(\eta^\sigma(T)_{\mathsf{pre}(t)}) & \textit{otherwise.} \end{cases} \tag{4.11}$$

We say that the subtree-embedding $\eta^\sigma(T)$ encounters the EXIT *if it contains a vertex labeled* EXIT *and $\eta^\sigma(T)$ encounters a cycle if it contains two vertices having the same label.*

Figure 4.3 illustrates an example of a subtree of $\mathcal{T}$ and the corresponding subtree-embedding in $\mathcal{G}$. For any tree T specified in Definition 76, the root of T will always be EMPTYADDRESS so the root of $\eta^\sigma(T)$ will always be ENTRANCE. The subtree-embedding $\eta^\sigma(T)$ of a tree T will correspond to the subgraph of $\mathcal{G}^\sigma$ that contain vertices which can be reached by following the addresses given by vertex labels of T in $\mathcal{G}^\sigma$ beginning at the ENTRANCE.

Next, we will show that for a fixed sub-tree of $\mathcal{T}$ and a randomly chosen color-preserving permutation σ, it is not possible, except for exponentially small probability, for the corresponding



(a) The subtree T of $\mathcal{T}$ (b) The subtree-embedding $\eta^\sigma(T)$ of T in $\mathcal{G}$

Figure 4.3: Example of a subtree-embedding for the graph $\mathcal{G}$ in Figure 2.1 and the identity permutation σ.

subtree-embedding to contain the EXIT or a path that forms a cycle in $\mathcal{G}^\sigma$.

Lemma 77. *Let T be a subtree of the address tree $\mathcal{T}$ of size $p(n)$ that contains the vertex labeled* EMPTYADDRESS *but does not contain vertices having labels in* $\mathsf{SpecialAddresses} \setminus \{\text{EMPTYADDRESS}\}$. *Let the permutation σ be chosen according to the distribution D_n. Then, the probability that the subtree-embedding $\eta^\sigma(T)$ encounters the* EXIT *or a cycle is at most $4p(n)^4 2^{-n/3}$.*

Proof. Suppose that the subtree-embedding $\eta^\sigma(T)$ contains a vertex v labeled EXIT. Let t denote the label of the vertex of T corresponding to v. Then, the path-embedding $\eta^\sigma(t)$ encounters the EXIT. Therefore, since T has at most $p(n)$ vertices, the probability of $\eta^\sigma(T)$ encountering the EXIT is at most $p(n) \cdot 4p(n)^2 2^{-n/3} = 4p(n)^3 2^{-n/3}$ by Corollary 75.

Now, suppose that the subtree-embedding $\eta^\sigma(T)$ encounters a cycle. That is, it contains two vertices v_1 and v_2 having the same label. Without loss of generality, assume that the respective parents u_1 and u_2 of v_1 and v_2 in T (if they exist) does not have the same labels; otherwise, re-label v_1 as u_1 and v_2 as u_2. Let t_1 and t_2 be the labels of the vertices of T corresponding to v_1 and v_2 respectively. Let $t_{1,2}$ denote the sequence resulting from the concatenation of t_1 with the reverse of t_2. By our above assumption, no color can appear consecutively in $t_{1,2}$ so $t_{1,2}$ does not contain an even-length palindrome. Thus, starting from the ENTRANCE and following the sequence of

colors given by $t_{1,2}$ in $\mathcal{G}^\sigma$ will result in reaching back the ENTRANCE without backtracking. This means that the path-embedding $\eta^\sigma(t_{1,2})$ encounters a cycle in $\mathcal{G}^\sigma$. Therefore, since there are at most $\binom{p(n)}{2}$ pairs of vertices of T, the probability of $\eta^\sigma(T)$ encountering a cycle is at most $\binom{p(n)}{2} \cdot 4p(n)^2 2^{-n/3} \leq 2p(n)^4 2^{-n/3}$ by Corollary 75.

We obtain the desired result by union bounding over the probabilities specified above. □

We now establish that for any uniformly random permutation σ and any classical algorithm that samples from the subtrees of $\mathcal{T}$, we cannot hope to find the EXIT or a cycle in $\mathcal{G}$ with more than exponentially small probability.

Lemma 78. *Let the permutation σ be chosen according to the distribution D_n. Let $\mathcal{S}$ be the set of subtrees of the address tree $\mathcal{T}$ of size $p(n)$ that contains the vertex labeled* EMPTYADDRESS *but does not the contain vertices having labels in* $\mathsf{SpecialAddresses} \setminus \{\text{EMPTYADDRESS}\}$.[3] *Let $\mathcal{A}_{\mathsf{classical}}$ be a classical query algorithm that samples a tree T from $\mathcal{S}$ and computes the subtree-embedding $\eta^\sigma(T)$. Then, the probability that $\mathcal{A}_{\mathsf{classical}}$ finds the* EXIT *or a cycle in $\mathcal{G}$ is at most $4p(n)^4 2^{-n/3}$.*

Proof. The algorithm $\mathcal{A}_{\mathsf{classical}}$ finds the EXIT if the subtree-embedding $\eta^\sigma(T)$ contains it. On the other hand, since $\eta^\sigma(T)$ corresponds to a connected subgraph of $\mathcal{G}^\sigma$, $\mathcal{A}_{\mathsf{classical}}$ finds a cycle in $\mathcal{G}^\sigma$ if there are two vertices in the tree $\eta^\sigma(T)$ that have the same label. Therefore, this Lemma is a direct consequence of Lemma 77 and convexity. □

We conclude this chapter with our main result about the existence of a distribution for which it is hard for a natural class of classical algorithms to find the EXIT or a cycle in the Welded Tree Graph sampled according to this distribution.

Theorem 79. *Let $\mathcal{S}$ be the set of subtrees of the address tree $\mathcal{T}$ of size $p(n)$ that contains the vertex labeled* EMPTYADDRESS *but does not the contain vertices having labels in* $\mathsf{SpecialAddresses} \setminus \{\text{EMPTYADDRESS}\}$.

[3] Recall that $\mathcal{T}$ can be computed using 2 queries to the classical oracle η^σ.

Then, there exists a distribution $\mathcal{D}_n$ over size n 3-colored Welded Tree Graphs such that for any classical query algorithm $\mathcal{A}_{\text{classical}}$ that samples a tree T from $\mathcal{S}$ and computes the associated subtree-embedding in $\mathcal{G}'$, the probability that $\mathcal{A}_{\text{classical}}$ finds the EXIT *or a cycle in $\mathcal{G}'$ is at most $4p(n)^4 2^{-n/3}$ where $\mathcal{G}'$ is the Welded Tree Graph sampled from $\mathcal{D}_n$.*

Proof. Consider the distribution $\mathcal{D}_n$ specified by the following sampling process: choose σ according to the distribution D_n, and output $\mathcal{G}^\sigma$. From Lemma 78, we know that $\mathcal{D}_n$ satisfies the requirement of this theorem. □

Chapter 5: Closing remarks

We will finish by summarizing this thesis, discuss some interpretative implications and some relevant open questions.

5.1 Conclusion

In this work, we have shown that any quantum algorithm that is comprised of the gate set given in Definition 9 and whose state is always a superposition of computational basis states corresponding to a connected subgraph of the input Welded Tree (more precisely defined as Rooted algorithms in Definition 12) cannot find the EXIT (or a path to the EXIT) using polynomially many queries with more than inverse exponential probability. Even though we do not rule out the existence of efficient path-finding quantum algorithms, it would be hard to imagine any such algorithm to critically use information stored in vertex labels, or to 'forget' the ENTRANCE and later compute a path to the ENTRANCE along with computing a path to the EXIT. Our strategy is to classically approximately simulate any such quantum algorithm in Chapter 3 and show that any classical algorithm of a certain type that includes our simulation algorithm cannot find the EXIT or a cycle (with respect to 3-color Welded Tree Oracle) in Chapter 4. In particular, we show that this classical simulation (stated in Algorithm 3) exactly simulates the portion of the state of the given genuine rooted quantum algorithm that does not find the EXIT or go through a cycle in Section 3.5, and the portion of the state of the given algorithm that encounters the EXIT or a cycle is small in Section 3.6.

5.2 Implications

We conjecture that no efficient quantum query algorithm can find an ENTRANCE-to-EXIT path in a Welded Tree graph, which we proved for a natural class of quantum algorithms. If this conjecture is true, it will manifest some novel computational implications of quantum mechanics and strengthen our understanding of quantum computing as being qualitatively different from its classical counterpart. We shed light on some of these as follows.

1. In Chapter 1, we referred to the the simultaneous existence of an efficient EXIT-finding quantum algorithm and the non-existence of any path-finding genuine, rooted quantum algorithm as the computational analog of the multi-slit experiment. In the multi-slit experiment, when a particle is released from a source, it goes through many branches of superposition to hit a particular point on the screen but does not maintain any particular course of its journey. If we start placing detectors in this experiment, we destroy useful superpositions, which will result in the particle not being able to reach its destination on the screen. Analogously, the quantum algorithm of [1] for EXIT-finding traverses exponentially many ENTRANCE-to-EXIT paths without remembering a particular such path. But if we enforce this genuine quantum algorithm to be rooted, it cannot even reach the EXIT. Even though many results in quantum computing reflect the nature of quantum mechanics, this phenomena of quantum mechanics has never been as explicitly demonstrated earlier.

2. Usually, classical computers can search a certificate of a decision problem by computing polynomially many instances of this decision problem. But for quantum computers, it may not be the case. If quantum computers can efficiently search for a certificate, it might help classical machines verify the behaviour of untrusted quantum devices efficiently. Classically, it not possible to find the EXIT without traversing through an ENTRANCE–EXIT path. Therefore,

if our above conjecture is true, it would demonstrate that solving a decision problem and searching for a certificate of it's decision are qualitatively different problems for quantum computers. It would also mean that, for certain decision problems, quantum computers can efficiently only determine information that cannot be used by classical computers to efficiently verify a solution.

5.3 Open questions

Even though the result in this thesis makes partial progress towards resolving an open problem, many questions of interest still remain open. We highlight some of them as follows.

1. The most immediate open question is whether it is possible to generalize our results to show quantum hardness of Welded Tree path-finding problem (see Definition 5) for all efficient quantum query algorithms.

   Is it possible to do away with the assumption of genuine-ness (Definition 10)? We believe that the assumption of being genuine is more natural and would be easier to remove as a non-genuine algorithm would involve gates that might use the information about vertex labels in a critical way, which are chosen randomly and independently from the structure of the given Welded Tree Graph. Therefore, we expect a non-genuine algorithm to be (approximately) simulated by a genuine algorithm.

   On the other hand, can we remove the assumption of rooted-ness (Definition 12)? Even though one might expect an algorithm for path-finding to remember its path to the ENTRANCE, it is not clear whether an algorithm can 'forget' the path to the ENTRANCE before re-computing it. It would be very intriguing to investigate whether there is an efficient algorithm that temporarily detaches from the ENTRANCE during its course, or this possibility could be ruled out.

2. In Chapter 4 (specifically, Theorem 79), we showed that for the 3-color Welded Tree Oracle, any classical algorithm sampling from any distribution of polynomial-size subtrees of the address tree (see Definition 24) and querying according to the addresses given by the labels of vertices of the sampled tree cannot find the EXIT or a cycle efficiently. Note that it is certainly possible for a classical algorithm to not behave in this particular way. We did not require proving hardness of EXIT-finding for the class of all classical query algorithms with respect to the 3-color Welded Tree Oracle since the special case of this statement that we showed is sufficient for our purposes (i.e. to show Lemma 55). However, we conjecture that such an assertion would be true and proving it would be of independent interest.

3. Can quantum computers output enough information that can be used by classical machines to efficiently verify their behaviour? Quantum Prover Interactive Protocol (**QPIP**) informally refers to the complexity class with a **BQP** prover and a **BPP** verifier. It is not known whether **BQP** is contained in **QPIP** with one round. If there is no efficient quantum query algorithm for path-finding, then is it possible to construct a variant of the Welded Tree problem that shows a relativized separation between **BQP** and one round **QPIP**?

   Consider the graph that is formed by taking two copies of a Welded Tree Graph (with a different ENTRANCE and EXIT label for each copy) and joining the ENTRANCE of one with the ENTRANCE of the other and the EXIT of one with the EXIT of the other. Given oracle access to this graph and labels of the ENTRANCE of one of the copies and the EXIT of both the copies (without revealing their identity), our problem requires determining which EXIT label corresponds to the given ENTRANCE label. Using the algorithm of [1], it is easy to see that this problem is contained in **BQP**. However, if finding an ENTRANCE–EXIT is hard in this graph and all the positive certificates encode some ENTRANCE–EXIT path, then this problem might not admit a one round **QPIP** protocol.

Bibliography

[1] Andrew M. Childs, Richard Cleve, Enrico Deotto, Edward Farhi, Sam Gutmann, and Daniel A. Spielman. Exponential algorithmic speedup by a quantum walk. In *Proceedings of the 35th Annual ACM Symposium on Theory of Computing, June 9-11, 2003, San Diego, CA, USA*, pages 59–68, New York, NY, USA, 2003. Association for Computing Machinery. doi: 10.1145/780542.780552.

[2] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, 26(5):1484–1509, 1997. Preliminary version in FOCS 1994.

[3] Sean Hallgren. Polynomial-time quantum algorithms for Pell's equation and the principal ideal problem. *Journal of the ACM*, 54(1):article 4, 2007. doi: 10.1145/1206035.1206039. Preliminary version in STOC 2002.

[4] Kiran S. Kedlaya. Quantum computation of zeta functions of curves. *Computational Complexity*, 15(1):1–19, 2006. doi: 10.1007/s00037-006-0204-7.

[5] Kirsten Eisenträger, Sean Hallgren, Alexei Kitaev, and Fang Song. A quantum algorithm for computing the unit group of an arbitrary degree number field. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing*, pages 293–302, 2014. doi: 10.1145/2591796.2591860.

[6] Seth Lloyd. Universal quantum simulators. *Science*, 273(5278):1073–1078, 1996. doi: 10.1126/science.273.5278.1073.

[7] Matthew Coudron and Sanketh Menda. Computations with greater quantum depth are strictly more powerful (relative to an oracle). In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, June 22-26, 2020, Chicago, IL, USA*, page 889–901, New York, NY, USA, 2020. Association for Computing Machinery. doi: 10.1145/3357713.3384269.

[8] Richard Cleve and John Watrous. Fast parallel circuits for the quantum Fourier transform. In *Proceedings of the 41st IEEE Symposium on Foundations of Computer Science*, pages 526–536, 2000. doi: 10.1109/SFCS.2000.892140.

[9] Nai-Hui Chia, Kai-Min Chung, and Ching-Yi Lai. On the need for large quantum depth. In *Proceedings of the 52nd Annual ACM Symposium on Theory of Computing*, page 902–915, 2020. doi: https://doi.org/10.1145/3357713.3384291.

[10] Scott Aaronson. Open problems related to quantum query complexity. *ACM Transactions on Quantum Computing*, 2(4), 2021. doi: 10.1145/3488559.

[11] Ansis Rosmanis. Quantum snake walk on graphs. *Physical Review A*, 83(2):022304, 2011. doi: 10.1103/PhysRevA.83.022304.

[12] Daniel R. Simon. On the power of quantum computation. *SIAM J. Comput.*, 26(5):1474–1483, 1997. doi: 10.1137/S0097539796298637.