Quantum Algorithms and the Power of Forgetting

Andrew M. Childs
Joint Center for Quantum Information and Computer Science, College Park, MD, USA
Department of Computer Science, University of Maryland, College Park, MD, USA
Institute for Advanced Computer Studies, University of Maryland, College Park, MD, USA

Matthew Coudron
Joint Center for Quantum Information and Computer Science, College Park, MD, USA
National Institute of Standards and Technology, Gaithersburg, MD, USA
Department of Computer Science, University of Maryland, College Park, MD, USA

Amin Shiraz Gilani
Joint Center for Quantum Information and Computer Science, College Park, MD, USA
Department of Computer Science, University of Maryland, College Park, MD, USA

Abstract
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 [3]. 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. 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 unclear how an algorithm could benefit by deviating from this behavior. Our no-go result suggests that, for some problems, quantum algorithms must necessarily forget the path they take to reach a solution in order to outperform classical computation.

2012 ACM Subject Classification Theory of computation → Quantum computation theory

Keywords and phrases Quantum algorithms, quantum query complexity

Digital Object Identifier 10.4230/LIPIcs.ITCS.2023.37


Funding Andrew M. Childs: Army Research Office (grant W911NF-20-1-0015); National Science Foundation (grant CCF-1813814); and Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, Accelerated Research in Quantum Computing program. Matthew Coudron: National Institute of Standards and Technology. Contributions to this work by NIST, an agency of the US government, are not subject to US copyright. Amin Shiraz Gilani: United States Educational Foundation in Pakistan and the Institute of International Education (Fulbright scholarship).

Acknowledgements We thank Matt Kovacs-Deak for many helpful discussions.

1 Introduction
Quantum algorithms use interference of many branches of a superposition to solve problems faster than is possible classically. Shor’s factoring algorithm [13] 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 (e.g., [7, 8, 9]) provide a
speedup by similarly detecting periodic structures. While a few other examples of dramatic quantum speedup are known – notably including the simulation of quantum dynamics [11] – 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 can 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 [3]. 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 [6], whereas all previously known exponential quantum speedups only require logarithmic quantum depth, including Shor’s factoring algorithm [5]. (The only other known computational problem exhibiting an exponential quantum speedup, yet requiring polynomial quantum depth, was recently constructed in [2].)

The welded tree problem is defined on a “welded tree graph” that is formed by joining the leaves of two binary trees with a cycle that alternates between them, as shown in Figure 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 – so the quantum walk achieves exponential speedup.

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 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 well-known 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 already arose in the original paper on the welded tree problem [3] and has remained open since, recently being highlighted in a survey of Aaronson [1].

In one attempt to solve this problem, Rosmanis studied a model of “snake walks”, which allow extended objects to move in superposition through graphs [12]. 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. Although 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 impossibility 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 classically verifiable certificate for the solution once they have solved the problem. 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 information.

In this paper, we take a step toward showing hardness of the welded tree path-finding problem. Specifically, we show hardness under two natural assumptions that we formalize in Section 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 [3] and the snake walk [12] can be implemented by genuine algorithms. It is hard to imagine how non-genuine algorithms could gain an advantage over genuine algorithms, but we leave further exploration of this topic for future work.

We also assume that the algorithm is rooted. Informally, a rooted algorithm is one that always maintains (i.e., remembers) a path from the entrance to every vertex appearing in its state. Note that the exit-finding algorithm of [3] is crucially not rooted. Nonetheless, it is natural to focus on rooted algorithms when considering the path-finding problem, since a non-rooted path-finding algorithm would effectively have to detach from the entrance and later find it again. While we cannot rule out this possibility, it seems implausible. Although remembering a path to the entrance limits how quantum interference can occur, it does not eliminate interference entirely – in fact, rooted algorithms can exhibit exponential constructive and destructive interference. Furthermore, if the snake walk [12] were to find a path from entrance to exit, it would most naturally do so in a rooted fashion.

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 establish this, we show that for any genuine, rooted quantum query algorithm $A$, there is a

---

1 For example, in Simon’s algorithm [14], we learn the hidden string and can easily find collisions. In Shor’s algorithm [13], the factors reveal the structure of the input number and their correctness can be easily checked.
classical query algorithm using at most polynomially more queries that can approximately sample from the output state of \( A \) (measured in the computational basis) up to a certain error term \( |\psi_{ugly}\rangle \) (defined in Section 4.4). This error term can be intuitively described as the part of the superposition of rooted configurations that has ever encountered a cycle or the exit in the welded tree graph during the entire course of the algorithm. Because elements of a quantum superposition need not have a well-defined classical history, the precise definition of our error term is somewhat involved. Nonetheless, we are able to bound the sampling error of our efficient classical simulation of \( A \) using an inductive argument.

We construct the classical simulation as follows. First, using exponential time and only a constant number of classical queries, the algorithm processes the circuit diagram of the genuine, rooted quantum algorithm and samples a “transcript” (defined in Section 3) that describes a computational path the quantum algorithm could have taken, neglecting the possibility of encountering a cycle or the exit. 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 Section 4 that this efficient classical query algorithm is almost as likely to find an entrance–exit path as the original genuine, rooted quantum algorithm. We do this by showing that the classical algorithm exactly simulates the part of the quantum state that does not encounter a cycle or the exit, and that the remaining error term, \( |\psi_{ugly}\rangle \), is exponentially small. A major technical challenge is that our classical hardness result (shown in Section 5) does not immediately show that \( |\psi_{ugly}\rangle \) is small as it may be possible for a quantum algorithm to foil a classical simulation by computing and uncomputing a cycle, and “pretending” to have never computed it. We overcome this issue by considering the portions of the state that encounter a cycle at each step and inductively bounding their total mass.

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 the lower bound of \([3]\).\(^2\) This alternative coloring scheme substantially reduces the complexity of the analysis in Sections 2–4. 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 \( t \) 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 15) 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.

However, our choice of the 3-coloring model comes at the cost of having to redesign the proof of classical hardness of finding the exit vertex in the welded tree graph. The original classical hardness proof \([3]\) 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 \([3]\). In contrast, a valid 3-coloring of the welded tree graph does not have this “local re-colorability” property: changing a single edge color might require a global

\(^2\) 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 \([3]\).
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 Section 5, which may also be of independent interest.

Note that our hardness result for exit-finding in the 3-color model implies the hardness result for exit-finding in the colorless model of [3] (which is equivalent to a restricted class of locally-constructible 9-colorings), but not the other way around. This is because, a priori, the given 3-coloring could leak global information about the graph that the 9-coloring does not. On the other hand, our hardness result for exit-finding in the 3-color model combined with the analysis of Section 2 implies hardness of path finding for genuine, rooted algorithms in both the 3-color and 9-color models, as well as the colorless model.

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.

Due to space limitations, no proofs are included in this condensed version of the paper. For a full technical version including all proof details, see Ref. [4].

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.

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” processing of vertex labels, 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 1 (Welded tree). A graph $G_n$ 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). Each vertex in $G_n$ is labeled by a $2n$-bit string.

Henceforth, we refer to the input welded tree graph of size $n$ as $G$. We use $V_G$ to denote the set of vertices of $G$. Since $G$ is bipartite and each vertex $v \in V_G$ has degree at most 3, $G$ can be edge-colored using only 3 colors [10]. Therefore, we suppose that the edges of $G$ are colored from the set $C := \{\text{red, green, blue}\}$. We define a classical oracle function $\eta_c : \{0, 1\}^{2n} \rightarrow \{0, 1\}^{2n}$ that encodes the edges of color $c \in C$ in $G$. 
We now describe the spaces that our algorithms act on.

Definition 4 (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.

2.1 Genuine algorithms

We now precisely describe the set of gates that we allow quantum query algorithms to employ for querying and manipulating the vertex labels in a meaningful way.

Definition 5 (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 $c \in C$ where the control qubit is in the workspace, and $O_c$ acts on the $j$th and $k$th vertex registers for some distinct $j, k \in [p(n)] := \{1, \ldots, p(n)\}$ as $O_c : |v_j⟩|v_k⟩ \mapsto |v_j⟩|v_k \oplus \eta_c(v_j)⟩$ where $\eta_c$ is specified in Definition 2.

2. Controlled-$e^{iθT}$ rotations for any $θ \in [0, 2π)$ where the control qubit is in the workspace and the Hamiltonian $T$ is defined, similarly to [3], to act on the $j$th and $k$th vertex registers for some distinct $j, k \in [p(n)]$ as $T : |v_j⟩|v_k⟩ \mapsto |v_j⟩|v_k⟩$. As per part 1, $Λ(e^{iθT})$ denotes the controlled-$e^{iθT}$ gate.

3. Equality check gates $E_j$, 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 $E_j : |v_j⟩|v_k⟩|w_a⟩ \mapsto |v_j⟩|v_k⟩|w_a \oplus δ[v_j = v_k]⟩$ where $δ[P]$ is 1 if $P$ is true and 0 if $P$ is false.

4. NOEDGE check gates $N_j$, 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 $N_j : |v_j⟩|w_a⟩ \mapsto |v_j⟩|w_a \oplus δ[v_j = \text{NOEDGE}]⟩$.

5. ZERO check gates $Z_j$, 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 $Z_j : |v_j⟩|w_a⟩ \mapsto |v_j⟩|w_a \oplus δ[v_j = 0^{2n}]⟩$.

6. Arbitrary two-qubit gates (or, equivalently, arbitrary unitary transformations) restricted to the workspace register.
We now define the notion of genuine algorithms using Definition 5. Let \( O = \{ O_c : c \in C \} \) denote a particular randomly selected welded tree oracle, and let \( A(O) \) denote a quantum algorithm that makes quantum queries to \( O \).

**Definition 6 (Genuine algorithm).** We call a quantum query algorithm \( A \) genuine if, for the given welded tree oracle \( O \), \( A(O) \) acts on the vertex space and the workspace as follows.

1. \( A(O) \) begins with an initial state \( |\psi_{\text{initial}}\rangle = |\text{ENTRANCE}\rangle \otimes \left( |0^{2n}\rangle \right)^{\otimes (p(n)-1)} \otimes |0\rangle_{\text{workspace}} \).

2. Then, it applies a \( p(n) \)-gate genuine circuit \( C \) (as in Definition 5) on \( |\psi_{\text{initial}}\rangle \) to get the state \( |\psi_A\rangle \).

3. Finally, it measures all the vertex registers of \( |\psi_A\rangle \) in the computational basis and outputs the corresponding vertex labels.

We focus on genuine algorithms because they are easier to analyze than fully general algorithms, but do not seem to eliminate features that would be useful in a path-finding algorithm. Genuine algorithms are only restricted in the sense that they cannot use vertex labels other than by storing them, acting on them with the input or output register of an oracle gate, performing phased swaps of the vertex label positions, and checking whether they are equal to zero or NOEDGE. Since the vertex labels are arbitrary and uncorrelated with the structure of the welded tree, it is hard to imagine how a general quantum algorithm could gain an advantage over genuine quantum algorithms by using the vertex labels in any other way.\(^3\) Thus, genuine algorithms describe a natural class of strategies that should offer insight into the more general case.

We also emphasize that the only proposed algorithms for the welded tree problem (and the associated path-finding problem) are genuine. The only such algorithms we are aware of are the exit-finding algorithm of \([3]\) and the snake-walk algorithm analyzed by Rosmanis \([12]\).

Intuitively, the exit-finding algorithm of \([3]\) is genuine since it performs a quantum walk on the welded tree graph, and such a process does not depend on the vertex labels. More concretely, a close inspection of the exit-finding algorithm of \([3]\) reveals that every gate in the algorithm is an allowed gate in Definition 5 (even with the above minor modification). This means that the algorithm is genuine as per Definition 6. (As a technical aside, note that the algorithm works with any valid coloring of the welded tree graph, so in particular, it works for our chosen 3-color model by simply limiting the set of colors in the algorithm.)

Similarly, in \([12]\), Rosmanis defines a quantum snake walk algorithm on a particular welded tree graph \( G \) to be a quantum walk on a corresponding “snake graph” \( G_n \), which has one vertex for each distinct “snake” of length \( \ell \) in \( G \). Here a “snake” of length \( \ell \) refers to a length-\( \ell \) vector of consecutive vertices of \( G \). Although it is more complicated to decompose this algorithm into the gates of Definition 5, this can be done, showing that the snake walk algorithm is genuine. Furthermore, it is intuitive that the snake walk algorithm should not depend on the specific vertex labels simply because it is defined to be a quantum walk on \( G_n \), a graph whose connectivity does not depend on the vertex labels of \( G \).

### 2.2 Rooted algorithms

We now define the notion of a rooted algorithm. Intuitively, a state in the vertex space is rooted if it corresponds to a set of labels of vertices from \( V_G \) (and the NOEDGE and \( 0^{2n} \) labels) that form a connected subgraph containing the ENTRANCE (neglecting the NOEDGE and \( 0^{2n} \) labels, if present).

\(^3\) The proof that classical algorithms cannot efficiently find the exit effectively shows that classical algorithms cannot benefit from non-genuine behavior \([3]\). While it seems harder to make this rigorous for quantum algorithms, similar intuition holds.
Definition 7 (Rooted state). We say that a computational basis state $|\psi\rangle$ in the vertex space is rooted if \textsc{entrance} is stored in some vertex register of $|\psi\rangle$ and, for any vertex label $v$ stored in any of the vertex registers of $|\psi\rangle$, $v \in \mathcal{V}_G \cup \{0^{2n}, \text{noedge}\}$, and if $v \neq 0^{2n}$, 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 $c \in \mathcal{C}$.

We say that an algorithm is rooted if all its intermediate states are superpositions of rooted states.

Definition 8 (Rooted algorithm). A quantum query algorithm $A$ is rooted if, for the given welded tree oracle $O$, at each intermediate step of $A(O)$, every computational basis state in the support of the vertex space of the quantum state maintained by $A$ is rooted.

Non-rooted behavior can be useful for exploring the welded tree graph. In particular, the algorithm of [3] for finding the \textsc{exit} is not rooted, since it only maintains a single vertex (in superposition). However, a path-finding algorithm must store information about many vertices, and the value of detaching from the \textsc{entrance} is unclear since the algorithm must ultimately reattach. Note that the snake walk [12] is initially rooted, the most natural way for it to find a path from \textsc{entrance} to \textsc{exit} is arguably to do so while remaining rooted, though the algorithm may become non-rooted if it is run for long enough.

3 Transcript states

For any genuine quantum query algorithm $A$ that makes $p(n)$ oracle queries to the oracle $O$ of the input welded tree $G$, we associate a quantum state $|\phi_A\rangle$, which we call the transcript state of $A(O)$. As we will see in Definition 15 below, instead of storing the label of a vertex $v$, the transcript state $|\phi_A\rangle$ stores a path from the \textsc{entrance} to $v$. We refer to this path as the address of $v$, which we now define.

Definition 9 (Vertex addresses). We say that a tuple $t$ of colors from $\mathcal{C}$ is an address of a vertex $v$ of $G$ if $v$ is reached by starting at the \textsc{entrance} and following the edge colors listed in $t$. For completeness, we assign special names \textsc{zeroaddress}, \textsc{noedgeaddress}, and \textsc{invalidaddress} to denote the addresses of vertex labels $0^{2n}$, \textsc{noedge}, and \textsc{invalid}, respectively. We denote the empty tuple by the special name \textsc{emptyaddress}. Let $\text{SpecialAddresses} := \{\text{zeroaddress}, \text{emptyaddress}, \text{noedgeaddress}, \text{invalidaddress}\}$. Finally, we define $\text{Addresses} := \text{SpecialAddresses} \cup \bigcup_{i \in [p(n)]} \mathcal{C}^i$ where $\mathcal{C}^i$ 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 $G$. 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 \textsc{entrance}.

Definition 10 (The missing 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 \textsc{entrance} in $G$. 
Definition 11 (Address tree). The address tree $\mathcal{T}$ (see Figure 2) is a binary tree of depth $p(n)$ with 3 additional vertices. Its vertices and edges are labeled by addresses and colors, respectively, as follows. The 3 additional vertices are labeled by each address in $\text{SpecialAddresses} \setminus \{\text{emptyaddress}\}$. The root of $\mathcal{T}$ is labeled by $\text{emptyaddress}$. It is joined to the vertex labeled $\text{noedgeaddress}$ by a directed edge of color $c_*$, and to 2 other vertices, each by an undirected edge of a distinct color from $C \setminus \{c_*\}$. For each color $c \in C$, the vertices labeled $\text{zeroaddress}$ and $\text{noedgeaddress}$ have a directed edge colored $c$ to the vertex labeled $\text{invalidaddress}$. The vertex labeled $\text{invalidaddress}$ also has 3 self-loop edges, each of a distinct color from $C$. Every other vertex in $\mathcal{T}$ is joined to 3 other vertices, each by an undirected edge of a distinct color from $C$. Every vertex $t$ of $\mathcal{T}$ whose label is not in $\text{SpecialAddresses}$ is labeled by the sequence of colors that specifies the (shortest) path from $\text{emptyaddress}$ 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 $\mathcal{T}$ contains an even-length palindrome.
- Beginning at the vertex labeled $\text{emptyaddress}$ and traversing any sequence of colors in $\mathcal{T}$ leads to some vertex of $\mathcal{T}$. Therefore, in the definition of the transcript state (Definition 15), and hence in the algorithm analyzed in Section 4, 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 $|\phi_A\rangle$ is the state that results from running the algorithm $A$ on the address tree $\mathcal{T}$ instead of the actual welded tree graph $\mathcal{G}$. If $A$ does not explore cycles in $\mathcal{G}$ to a significant extent, then $|\phi_A\rangle$ should be a good approximation of the state $|\psi_A\rangle$ produced by running $A$ on $\mathcal{G}$, as in Definition 6. In Section 4, we show that this is indeed the case for any genuine, rooted quantum algorithm $A$.

Figure 2 Address tree $\mathcal{T}$ of depth 3. For the sake of brevity, we have removed the suffix ADDRESS for all the addresses in $\text{SpecialAddresses}$ and the tuple brackets for all the addresses not in $\text{SpecialAddresses}$. Notice that, for each vertex, there is an edge (either directed or undirected) of each color outgoing from each vertex in $\mathcal{T}$.
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 $B$. In our analysis, the registers we consider can never contain any string that is not in the range of the $B$ mapping. Therefore, it is sufficient to define $B^{inv}$ over the range of $B$. Nevertheless, we define $B^{inv}$ over $\{0,1\}^{2p(n)}$ for the sake of completeness.

**Definition 12 (B mapping).** Let $\mathcal{V}_T$ denote the set of labels of vertices of the address tree $T$. Let $S$ be a subset of $\{0,1\}^{2p(n)}$ of size $|\mathcal{V}_T|$ containing $0^{2p(n)}$. Let emptystring, noedgestring, and invalidstring be any distinct fixed strings in $S \setminus \{0^{2p(n)}\}$. Then $B: \mathcal{V}_T \to S$ is a bijection mapping zeroaddress to $0^{2p(n)}$, emptyaddress to emptystring, noedgeaddress to noedgestring, and invalidaddress to invalidstring. Furthermore, the mapping $B^{inv}: \{0,1\}^{2p(n)} \to \mathcal{V}_T$ sends $s \in S$ to $B^{-1}(s)$ and $s \notin S$ to invalidaddress.

We now define analogs of the spaces introduced in Definitions 3 and 4 that our transcript state (Definition 15) lies in and that our classical simulation algorithm (Algorithm 2) acts on.

**Definition 13 (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 $T$. 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 14 (Address workspace and address workspace register).** An address workspace register is a single-qubit register that stores arbitrary ancillary states. We allow 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 observe that the projection of $|\psi_A\rangle$ on the workspace is the same as the projection of the transcript state on the address workspace in the subspace not containing the cycle or a cycle. We are now ready to state the definition of the transcript state $|\psi_A\rangle$ associated with the quantum state $|\phi_A\rangle$.

**Definition 15 (Transcript state).** Consider a $p(n)$-query genuine, rooted quantum algorithm $A$. Given a circuit $C$ that implements $A$, acting on the vertex space and the workspace, let $\hat{C}$ be the quantum circuit that acts on the address space and the address workspace, obtained by the following procedure:\footnote{Notice that the time complexity of this procedure is linear in the size of the circuit $C$.}

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 in the genuine algorithm (recall Definition 6) with the new initial state $\phi_{\text{initial}} := |\text{emptystring}\rangle \otimes \left(0^{2p(n)}\right) \otimes |0\rangle_{\text{address workspace}}$.

In parts 3–8 below, we describe gates that act on the address space analogously to how the gates in Definition 5 act on the vertex space. For any vertex $v \in \mathcal{V}_G$, 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 state is produced by the unitary operation that results by replacing each vertex-space gate in the quantum algorithm $A$ 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-$O_c$ (controlled on address workspace register $a$ and acting on address registers $j$ and $k$), where $O_c: |s_j⟩|s_k⟩ \mapsto |s_j⟩|s_k⟩ ⊕ B(\lambda_c(Binv(s_j)))$.

4. Replace any controlled-$e^{iθT}$ gate in $C$ (controlled on workspace register $a$ and acting on vertex registers $j$ and $k$) with a controlled-$e^{iθT}$ gate (controlled on address workspace register $a$ and acting on address registers $j$ and $k$), where $T: |s_j⟩|s_k⟩ \mapsto |s_k⟩|s_j⟩$.

5. Replace any equality check gate $E$ in $C$ (controlled on vertex registers $j$ and $k$ and acting on workspace register $a$) with $\tilde{E}$ (controlled on address registers $j$ and $k$ and acting on address workspace register $a$), where $\tilde{E}: |s_j⟩|s_k⟩|w_a⟩ \mapsto |s_j⟩|s_k⟩|w_a⟩ ⊕ δ[s_j = s_k]$.

6. Replace any NOEDGE-check gate $N$ in $C$ (controlled on vertex register $j$ and acting on workspace register $a$) with $\tilde{N}$ (controlled on address register $j$ and acting on address workspace register $a$), where $\tilde{N}: |s_j⟩|w_a⟩ \mapsto |s_j⟩|w_a⟩ ⊕ δ[s_j = \text{NOEDGESTRING}]$.

7. Replace any ZERO-check gate $Z$ in $C$ (controlled on vertex register $j$ and acting on workspace register $a$) with $\tilde{Z}$ (controlled on address register $j$ and acting on address workspace register $a$), where $\tilde{Z}: |s_j⟩|w_a⟩ \mapsto |s_j⟩|w_a⟩ ⊕ δ[s_j = 0^{p(n)}]$.

8. Leave gates acting on the workspace unchanged.

The transcript state $|φ_A⟩$ is obtained by applying the circuit $\tilde{C}$ to the string $\text{EMPTYSTRING} = B(\text{EMPTYADDRESS})$, together with $p(n) − 1$ ancilla address registers storing the string $0^{p(n)} = B(\text{ZEROADDRESS})$. In other words, $|φ_A⟩ := \tilde{C}|φ_{\text{initial}}⟩$.

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

## 4. Classical simulation of genuine, rooted algorithms

We now describe a classical algorithm for simulating genuine, rooted quantum algorithms. We begin in Section 4.1 by describing procedures for checking that the behavior of a quantum algorithm is genuine and rooted. While these procedures have no effect on a quantum algorithm with those properties, they enforce properties of the transcript state that are useful in our analysis. Then, in Section 4.2, we describe a mapping that sends states in the address space to states in the vertex space, which is used to describe our simulation algorithm in Section 4.3. In Section 4.4, we decompose the state into components that assist in our analysis. We show in Section 4.5 that the “good” part of the state of a genuine, rooted algorithm is related, via the mapping $L$ defined in Definition 22, to the “good” part of the state of our simulation at each intermediate step. Finally, we establish in Section 4.6 (using the result of Section 5) that no genuine, rooted quantum algorithm can find an ENTRANCE–EXIT path (or a cycle) with more than exponentially small probability.

### 4.1 Checking procedures

It is possible to efficiently check whether, for an oracle query $\forall(O_c)$ with the input vertex register storing $|v_j⟩$, the output vertex register contains $0^{2n}$ or $η_c(v_j)$.

**Remark 16 (Checking genuineness).** Given any genuine circuit $C$ with $|C|$ gates, one can efficiently construct a genuine circuit $C'$ consisting of $O(|C|)$ gates such that $C'$ has the same functionality as $C$ and verifies the condition stated in part 1 of Definition 5 before applying each oracle call $O_c$. Therefore, we assume without loss of generality that the given genuine circuit $C$ has built-in gadgets that verify this condition.

The consequence of Definition 15 and Remark 16 is a crucial observation about transcript states, which will turn out be useful in our analysis in Section 4.
Lemma 17. For any given rooted genuine circuit $C$, any string stored in any computational basis state in the support of the address space of the state $\tilde{C}|\phi_{\text{initial}}\rangle$ is in the range of $B$.

We can use a similar approach to efficiently modify a given quantum query algorithm to ensure that its state is always rooted. This modification will be useful for the analysis in Sections 4.4 and 4.5.

Remark 18 (Checking rootedness). Given any genuine circuit $C$ with $|C|$ gates, one can efficiently construct a rooted genuine circuit $C'$ consisting of $\text{poly}(|C|)$ gates such that $C'$ has the same functionality as $C$ and before applying each gate $G$, $C'$ checks whether the resulting state will remain rooted after the application of $G$. Therefore, we assume without loss of generality that the given rooted genuine circuit $C$ has built-in rootedness check gadgets.

Analogous to the notion of a rooted state defined in Definition 7, we define the notion of an address-rooted state as follows. Informally, a state in the address space is address rooted if, when it contains a string that encodes an address, it also contains the string that encodes its parent in $T$.

Definition 19 (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 $B^\text{inv}(s) \in V_T$ has a parent $t \neq \text{ZEROADDRESS}$, there exists a register of $|\phi\rangle$ that stores the string $B(t)$.

Using Remark 18, one can 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. This result substantially simplifies the analysis in Section 4.

Lemma 20. For any given rooted genuine circuit $C$, any computational basis state in the support of the address space of the state $\tilde{C}|\phi_{\text{initial}}\rangle$ is address rooted.

4.2 Mapping addresses to vertices

We now define an efficiently computable function $L'$ that maps an address $t$ to a corresponding vertex label $v$, and observe some relationships of addresses and the vertices they map to under $L'$. For $t \in \{\text{ZEROADDRESS, INVALIDADDRESS}\}$, 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 1 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 $v \leftarrow$ 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 \(i\)th color.

An implication of the definition of \(L'\) in Algorithm 1 is the following critical lemma. Informally, it states that if an address \(t\) does not encode an entrance–exit path or a cycle in \(G\), then the \(c\)-neighbor of the vertex corresponding to \(t\) in \(G\) is the same as the vertex corresponding to the \(c\)-neighbor of \(t\) in \(T\).

Lemma 21. Let \(v\) be any vertex label, let \(t\) be an address of \(v\), and let \(c \in C\). Furthermore, if \(t \notin \text{SpecialAddresses}\), suppose that following the edge colors of \(t\) starting at the entrance does not result in reaching the exit or finding a cycle in \(G\). Then \(L'(\lambda_c(t)) = \eta_c(v)\).

4.3 The classical algorithm

We now describe our classical algorithm (Algorithm 2) for simulating genuine quantum algorithms. To state the algorithm, we introduce several definitions, beginning with a map based on the function \(L'\) defined in Section 4.2.

Definition 22. For any \(m \in [p(n)]\), the mapping \(L: \left(\{0,1\}^{2^p(n)}\right)^m \rightarrow \left(\{0,1\}^{2^n}\right)^m\) sends \(m\) address strings to \(m\) vertex labels by acting as \(L'B^\text{inv}\) on each of the \(m\) registers.

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 \((L_{\text{workspace}} \otimes L_{\text{vertex}})|\chi\rangle\), with the map acting as the identity on the workspace register and as \(L\) on the vertex register.

To describe and analyze Algorithm 2, we consider individual gates and sequences of consecutive gates from the genuine circuit \(C\) defined in Definition 6. For this purpose, we consider the following definition.

Definition 23. For any \(i \in [p(n)]\), let \(C_i\) denote the \(i\)th gate of the circuit \(C\) in Definition 6. 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 \(j\)th gate. That is, \(C_{i,j} := C_i \cdots C_{i+1}\). Similarly, using the circuit \(\tilde{C}\) constructed in Definition 15, we define \(\tilde{C}_i\) and \(\tilde{C}_{i,j}\) for each \(i, j \in [p(n)] \cup \{0\}\) with \(i < j\).

Note that \(C_{i,i} = I\) and \(C_{i-1,i} = C_i\) (and similarly, \(\tilde{C}_{i,i} = I\) and \(\tilde{C}_{i-1,i} = \tilde{C}_i\)) for all \(i \in [p(n)]\). We use these gates to define transcript states and states of the quantum algorithm for partial executions.

Definition 24. For each \(i \in [p(n)] \cup \{0\}\), let \(\phi^{(i)}_{A} := \tilde{C}_0\cdots C_i|\phi_{\text{initial}}\) be the transcript state for the quantum algorithm \(A\) restricted to the first \(i\) gates of \(\tilde{C}\). Similarly, let \(\psi^{(i)}_{A} := C_0\cdots C_i|\psi_{\text{initial}}\) denote the state of the quantum algorithm \(A\) restricted to the first \(i\) gates of \(C\).

In particular, the state \(\phi^{(p(n))}_{A}\) is the transcript state corresponding to the quantum state \(\psi^{(p(n))}_{A}\) introduced in Definition 6. Now consider the following classical query algorithm for finding a path from the entrance to the exit.
Algorithm 2  Classical query algorithm $\mathcal{C}(A(O))$.

1. for $i \in [p(n)]$ do
2.   Given the circuit diagram $C_{0,i}$, compute the transcript state $\ket{\phi^{(i)}_A}$ as per Definition 15.
3.   Sample a computational basis state $\ket{\phi^{(i)}}$ in the address space at random with probability $\|\bra{\phi^{(i)}} \phi^{(i)}_A \rangle \|^2$.
4.   Compute the computational basis state $L\ket{\phi^{(i)}}$ in the vertex space.
5.   Output the labels of the vertices in $L\ket{\phi^{(i)}}$.

Note that when $A$ is genuine and rooted, the output of Algorithm 2 must be a connected subgraph of $\mathcal{G}$ containing the entrance. Therefore, if the output of Algorithm 2 contains the exit, it must reveal an entrance-to-exit path. In the remainder of Section 4, we show that the output of Algorithm 2 contains the exit (or a cycle) with exponentially small probability.

4.4 The good, the bad, and the ugly

We now define states $\ket{\psi^{(i)}_{good}}$, $\ket{\psi^{(i)}_{bad}}$, and $\ket{\psi^{(i)}_{ugly}}$, which are components of the state $\ket{\psi^{(i)}_A}$.

Intuitively, $\ket{\psi^{(i)}_{good}}$ represents the portion of the state of the algorithm after $i$ steps that has never encountered the exit or a near-cycle (i.e., a subgraph that differs from a cycle by a single edge) at any point in its history. $\ket{\psi^{(i)}_{bad}}$ represents the portion of the state of the algorithm after $i$ steps that just encountered the exit or a near-cycle at the $i$th step, and $\ket{\psi^{(i)}_{ugly}}$ 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 25. We say that a computational basis state $\ket{\phi}$ in the address space is $\phi$-bad if the subgraph corresponding to $L\ket{\phi}$ contains the exit or is at most one edge away from containing a cycle. A computational basis state $\ket{\phi}$ in the address space is $\phi$-good if it is not $\phi$-bad, i.e., if $L\ket{\phi}$ does not contain the exit and is more than one edge away from containing a cycle.

Similarly, a computational basis state $\ket{\psi}$ in the vertex space is $\psi$-bad if the subgraph corresponding to $\ket{\psi}$ contains the exit or is at most one edge away from containing a cycle and is $\psi$-good if it is not $\psi$-bad.

Note that the map $L$ is used to define the notion of good and bad states in the address space, but is not used for the corresponding notions in the vertex space. The good and bad states span the good and bad subspaces, respectively.

Definition 26. The $\phi$-BAD subspace is $\phi$-BAD := $\text{span}\{\ket{\phi} : \ket{\phi}$ is a $\phi$-bad state $\}$. The $\phi$-GOOD subspace is $\phi$-GOOD := $\text{span}\{\ket{\phi} : \forall \ket{\phi'} \in \phi$-BAD, $\bra{\phi'} \ket{\phi} = 0 \} = \text{span}\{\ket{\phi} : \phi$ is a $\phi$-good state $\}$. Let $\Pi^{\phi}_{bad}$ and $\Pi^{\phi}_{good}$ denote the projectors onto $\phi$-BAD and $\phi$-GOOD, respectively. The subspaces $\psi$-BAD and $\psi$-GOOD, and the projectors $\Pi^{\psi}_{bad}$ and $\Pi^{\psi}_{good}$, are defined analogously.

Notice that $\Pi^{\phi}_{bad} \Pi^{\phi}_{good}$ is $\Pi^{\phi}_{good} \Pi^{\phi}_{bad}$ is $0$ and $\Pi^{\phi}_{bad} + \Pi^{\phi}_{good}$ is $I$. Similarly, $\Pi^{\psi}_{bad} \Pi^{\psi}_{good}$ is $\Pi^{\psi}_{good} \Pi^{\psi}_{bad}$ is $0$ and $\Pi^{\psi}_{bad} + \Pi^{\psi}_{good}$ is $I$. We now define the states $\ket{\psi^{(i)}_{good}}$, $\ket{\psi^{(i)}_{bad}}$, and $\ket{\psi^{(i)}_{ugly}}$ that were described informally above.
Definition 27. We define \( \phi^{(i)}_{\text{good}} = \Pi_i \phi^{(i)}_A \) where \( \Pi_i = \Pi_{\text{bad}} \left( \phi^{(i)}_A - C \phi^{(i-1)}_A \right) \) and \( \phi^{(i)}_{\text{bad}} = \Pi_{\text{bad}} \phi^{(i)}_A \) where \( \Pi_{\text{bad}} \phi^{(i)}_A \). Moreover, let \( |\phi^{(i)}_{\text{good}}\rangle \) and \( |\phi^{(i)}_{\text{ugly}}\rangle \) for each \( i \in \{p(n)\} \), we define \( |\psi^{(i)}_{\text{good}}\rangle \), \( |\psi^{(i)}_{\text{bad}}\rangle \), and \( |\psi^{(i)}_{\text{ugly}}\rangle \), \( |\psi_{\text{good}}\rangle \), and \( |\psi_{\text{ugly}}\rangle \) analogously (using \( C \) in lieu of \( C \)).

Based on the intuitive description of \( |\psi^{(i)}_{\text{good}}\rangle \) and \( |\psi^{(i)}_{\text{bad}}\rangle \) that we provided earlier, we anticipate that the size (as quantified by the total squared norm) of the portion of the state \( |\psi^{(i)}_A\rangle \) that never encountered the \( \text{EXIT} \) or a near-cycle at any point in its history, and the size of the respective portions of the state \( |\psi^{(i)}_A\rangle \) that encountered the \( \text{EXIT} \) or a cycle at the \( i \)th or earlier steps, to sum to the size of \( |\psi^{(i)}_A\rangle \). The following lemma formalizes this intuition.

Lemma 28. Let \( i \in [p(n)] \cup \{0\} \). Then \( \| |\psi^{(i)}_{\text{good}}\rangle \|^2 + \sum_{j \in [i]} \| |\psi^{(i)}_{\text{bad}}\rangle \|^2 = 1 \).

We conclude this section strengthening the observations made in Remark 18 and Lemma 20.

Lemma 29. Let \( i \in [p(n)] \cup \{0\} \). Then any computational basis state in the support of \( |\psi^{(i)}_{\text{good}}\rangle \) or \( |\psi^{(i)}_{\text{bad}}\rangle \) is rooted, and any computational basis state in the support of \( |\psi^{(i)}_{\text{ugly}}\rangle \) is address rooted.

4.5 Faithful simulation of the good part

As any subtree of \( G \) without the \( \text{EXIT} \) can be embedded in \( T \), one might expect that the size of the portion of the state \( |\psi^{(i)}_A\rangle \) that never encountered the \( \text{EXIT} \) or a near-cycle at any point in its history is the same as the size of the portion of the state \( |\phi^{(i)}_A\rangle \) that never encountered the \( \text{EXIT} \) or a near-cycle. We formally show this via a sequence of lemmas that culminate in Lemma 40. We restrict our attention to the good parts of the states \( |\psi^{(i)}_A\rangle \) and \( |\phi^{(i)}_A\rangle \) in this subsection, beginning with a useful decomposition of \( |\phi^{(i)}_{\text{good}}\rangle \).

Definition 30. We define an indexed expansion of \( |\phi^{(i)}_{\text{good}}\rangle \) in the computational basis, as follows. Write \( |\phi^{(i)}_{\text{good}}\rangle = \sum_{p,q} \alpha^{(i)}_{p,q} |q^{(i)}\rangle |\phi^{(i)}_p\rangle \), where each \( |\phi^{(i)}_p\rangle \) denotes a computational basis state in the vertex register, each \( q^{(i)} \) specifies a computational basis state in the workspace register, and each \( \alpha^{(i)}_{p,q} \) is an amplitude. Define \( T_{\text{good}}^{(i)} \) to be the set of all indices \( p \) appearing in the expansion of \( |\phi^{(i)}_{\text{good}}\rangle \) with any corresponding non-zero amplitude \( \alpha^{(i)}_{p,q} \).

Analogous to Definition 27, we define computational basis states \( |\psi^{(i)}_p\rangle \) from \( |\phi^{(i)}_p\rangle \), and hence from \( |\phi^{(i)}_A\rangle \) rather than \( |\psi^{(i)}_A\rangle \).

Definition 31. For \( i \in [p(n)] \cup \{0\} \) and \( p \in \mathcal{T}_{\text{good}}^{(i)} \), let \( |\psi^{(i)}_p\rangle := L |\phi^{(i)}_p\rangle \).

Notice that it is not immediate from this definition that \( |\psi^{(i)}_p\rangle \) is in the support of the part of the state \( |\psi^{(i)}_A\rangle \) that is in the vertex space. However, by the end of this section, we will show that indeed this is the case.
We now show that the mapping $L$, defined in Definition 22, is a bijection from the set of address-rooted states in $\phi^{(i)}_{\text{good}}$ to the set of address-rooted states in $L\phi^{(i)}_{\text{good}}$.

\begin{itemize}
  \item \textbf{Lemma 32.} Let $i \in [p(n)]$ and $p, p' \in \mathcal{P}^{(i)}_{\text{good}}$. Suppose that $\psi^{(i)}_p = \psi^{(i)}_{p'}$. Then $\phi^{(i)}_p = \phi^{(i)}_{p'}$.

  The next lemma forms a key ingredient of Lemma 39, 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 $L$.

  \begin{itemize}
    \item \textbf{Lemma 33.} Let $i \in [p(n)]$, let $p \in \mathcal{P}^{(i)}_{\text{good}}$, and let $q$ be any workspace index. Then $L\tilde{C}_i|q^{(i-1)}\rangle\phi^{(i-1)}_p = C_i|q^{(i-1)}\rangle\psi^{(i-1)}_p$.
  
  Notice that the non-oracle gates in Definition 5 do not produce any “new information” about vertex labels. Based on this intuition, one might expect that the portion of $\tilde{\psi}^{(i-1)}_A$ (respectively $\tilde{\psi}^{(i-1)}_{\text{great}}$) 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 $i$th step. We formalize this as follows.

  \begin{itemize}
    \item \textbf{Lemma 34.} Let $i \in [p(n)]$ and suppose that $C_i$ is a genuine non-oracle gate. Then $\phi^{(i)}_{\text{good}} = \tilde{C}_i|q^{(i-1)}\rangle\phi^{(i-1)}_p$ and $\psi^{(i)}_{\text{good}} = C_i|q^{(i-1)}\rangle\psi^{(i-1)}_p$.
  
  For the analysis of oracle gates, we now define a subset of $\mathcal{P}^{(i-1)}_{\text{great}}$ that contains indices corresponding to computational basis states in the address space that do not contain the exit or a cycle even after the application of an oracle gate at the $i$th step. Inspired by the decomposition in Definition 30, we then define the components $\tilde{\phi}^{(i-1)}_{\text{great}}$ and $\tilde{\psi}^{(i-1)}_{\text{great}}$ of $\phi^{(i-1)}_A$ and $\psi^{(i-1)}_A$, respectively.

  \begin{itemize}
    \item \textbf{Definition 35.} Let $i \in [p(n)]$. Suppose that $C_i = \wedge(O_c)$ for some $c \in \mathcal{C}$. Then, define $\mathcal{P}^{(i-1)}_{\text{great}} := \{p \in \mathcal{P}^{(i-1)}_{\text{good}} : \exists p' \in \mathcal{P}^{(i)}_{\text{good}}\text{ such that } \tilde{C}_i|q^{(i-1)}\rangle\phi^{(i-1)}_{p'} = |q^{(i-1)}\rangle\phi^{(i-1)}_p\}$. Also, let $\tilde{\phi}^{(i-1)}_{\text{great}} := \sum_{p \in \mathcal{P}^{(i-1)}_{\text{great}}} \alpha_{p,q}^{(i-1)}|q^{(i-1)}\rangle\phi^{(i-1)}_p$ and $\tilde{\psi}^{(i-1)}_{\text{great}} := \sum_{p \in \mathcal{P}^{(i-1)}_{\text{great}}} \alpha_{p,q}^{(i-1)}|q^{(i-1)}\rangle\psi^{(i-1)}_p$.
  
  In the following lemma, we show that the $L$ mapping preserves the relationship between computational basis states in the support of $\phi^{(i-1)}_{\text{good}}$ and $\psi^{(i-1)}_{\text{good}}$: applying the oracle gate to a computational basis state in the support of $\phi^{(i-1)}_{\text{good}}$ results in a computational basis state in the support of $\phi^{(i-1)}_{\text{good}}$ exactly when applying the oracle gate to a computational basis state in the support of $L\phi^{(i-1)}_{\text{good}}$ results in a computational basis state in the support of $L\phi^{(i-1)}_{\text{good}}$.

  \begin{itemize}
    \item \textbf{Lemma 36.} Let $i \in [p(n)]$, $p \in \mathcal{P}^{(i-1)}_{\text{good}}$, and $p' \in \mathcal{P}^{(i)}_{\text{good}}$. Suppose that $C_i = \wedge(O_c)$ for some $c \in \mathcal{C}$. Then $\tilde{C}_i|q^{(i-1)}\rangle\phi^{(i-1)}_p = |q^{(i-1)}\rangle\phi^{(i-1)}_{p'}$ if and only if $C_i|q^{(i-1)}\rangle\psi^{(i-1)}_p = |q^{(i-1)}\rangle\psi^{(i-1)}_{p'}$.
  
  \end{itemize}
\end{itemize}
\end{itemize}

Definition 35 and Lemma 36 give rise to the following alternative definition of $\mathcal{P}^{(i-1)}_{\text{great}}$.

\begin{itemize}
  \item \textbf{Corollary 37.} Let $i \in [p(n)]$. Suppose that $C_i = \wedge(O_c)$ for some $c \in \mathcal{C}$. Then $\mathcal{P}^{(i-1)}_{\text{great}} = \{p \in \mathcal{P}^{(i-1)}_{\text{good}} : \exists p' \in \mathcal{P}^{(i)}_{\text{good}}\text{ such that } C_i|q^{(i-1)}\rangle\psi^{(i-1)}_p = |q^{(i-1)}\rangle\psi^{(i-1)}_{p'}\}$.
By the definition of $P_{\text{great}}^{(i-1)}$, there seems to be a bijective correspondence between elements of $P_{\text{great}}^{(i-1)}$ and $P_{\text{good}}^{(i-1)}$. We make this precise in the following lemma.

**Lemma 38.** Let $i \in [p(n)]$ and $c \in C$. Suppose that $C_i = \land (O_c)$ for some $c \in C$. Then $\phi_{\text{good}}^{(i)} = \tilde{C}_i \phi_{\text{great}}^{(i-1)}$. Moreover, $\psi_{\text{good}}^{(i)} = C_i \psi_{\text{great}}^{(i-1)}$ if $L \phi_{\text{great}}^{(i-1)} = \psi_{\text{great}}^{(i-1)}$.

The above analysis helps establish the following key lemma, which states that the states $\phi_{\text{good}}^{(i)}$ and $\psi_{\text{good}}^{(i)}$ are related by the mapping $L$. Intuitively, the oracle $O$ based on the address tree $T$ can faithfully simulate (modulo mapping $L$) the portion of the state $\psi_A^{(i)}$ of the algorithm $\mathcal{A}$ that does not encounter the exit or a cycle.

**Lemma 39.** For all $i \in [p(n)] \cup \{0\}$, $L \psi_{\text{good}}^{(i)} = \psi_{\text{good}}^{(i)}$.

Finally, we show the following relationship between the norms of $\phi_{\text{good}}^{(i)}$ and $\psi_{\text{good}}^{(i)}$, which will be very useful in bounding the probability of success of Algorithm $\mathcal{A}$ in Section 4.6.

**Lemma 40.** Let $i \in [p(n)] \cup \{0\}$. Then $\left\| \phi_{\text{good}}^{(i)} \right\| = \left\| \psi_{\text{good}}^{(i)} \right\|$.

### 4.6 The state is mostly good

In the remainder of this section, we conclude that it is hard for any rooted genuine quantum algorithm to find the exit (and hence, an entrance–exit path). We achieve this goal by bounding the mass of the quantum state $|\psi_A\rangle$ associated with any arbitrarily chosen rooted genuine quantum algorithm $\mathcal{A}$ that lies in the $\phi$-BAD subspace. We proceed by first using the result of Section 5 to bound the mass of the quantum state $|\phi_A^{(i)}\rangle$ that lies in the $\phi$-BAD subspace.

**Lemma 41.** Let $i \in [p(n)] \cup \{0\}$. Then $\left\| \Pi_{\text{bad}} \phi_A^{(i)} \right\|^2 \leq 4p(n)^4 \cdot 2^{-n/3}$.

From the result of Lemma 41, one might intuitively conjecture that the size of the portion of the state $|\phi_A^{(i)}\rangle$ after $i$ steps that encountered the exit or a near-cycle at some point in its history is small. We formalize this as follows.

**Lemma 42.** For all $i \in [p(n)] \cup \{0\}$, $\left\| \phi_{\text{ugly}}^{(i)} \right\| \leq 2ip(n)^2 \cdot 2^{-n/6}$.

The bound on the size of the portion of the state $|\phi_A^{(i)}\rangle$ after $i$ steps that never encountered the exit or a near-cycle directly follows from Lemma 42 as stated by the following corollary.

**Corollary 43.** Let $i \in [p(n)]$. Then $\left\| \phi_{\text{good}}^{(i)} \right\| \geq 1 - 2ip(n)^2 \cdot 2^{-n/6}$.

In the next lemma, we bound the mass of the portion of the state $|\psi_A^{(i)}\rangle$ after $i$ steps that encountered the exit or a near-cycle at some point in its history. This is a crucial lemma for our result in this section where we invoke Lemma 40 to deduce a statement about the quantum state of the genuine algorithm $\mathcal{A}$ using known properties of the state of our classical simulation of $\mathcal{A}$.

**Lemma 44.** Let $i \in [p(n)]$. Then $\left\| \psi_{\text{ugly}}^{(i)} \right\|^2 \leq 4i^2p(n)^2 \cdot 2^{-n/6}$.

The results in this section help establish our main theorem, which formally proves the hardness of finding an entrance–exit path for genuine, rooted quantum query algorithms.

**Theorem 45.** No genuine, rooted quantum query algorithm for the path-finding problem can find a path from entrance to exit with more than exponentially small probability.
5 Hardness of classical cycle finding with a 3-color oracle

In this section, 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 in Theorem 57 that the probability of finding the exit or a cycle for a natural class of classical algorithms is exponentially small even for a welded tree graph whose vertices are permuted according to the distribution $D_n$ specified in Definition 48 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 $G$ and randomizing the edges of the WELD (defined in Definition 46), making sure that the resulting graphs are valid 3-colored welded tree graphs.

The key ingredient of our analysis is Lemma 54, 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 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 54 with Lemma 8 of [3]. Indeed, the latter lemma shows that it is hard for any classical algorithm with access to a colorless welded tree oracle to satisfy either statement (i) or (ii) mentioned above. However, the argument of [3] is different than ours in two major ways: our proof is by induction, and we use randomness of the WELD and graph theoretic properties of any 3-coloring of a welded tree graph to argue the unlikeliness of statements (i) and (ii) while they use the hardness of guessing multiple coin tosses along with randomness of the WELD.

We begin by considering the following induced subgraphs of $G$.

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

Informally, $T_L$ and $T_R$ are induced subgraphs of $G$ on vertices in the left and right binary trees of $G$, respectively, while WELD is the induced subgraph of $G$ on the leaves of the left and right binary trees of $G$. Note that $T_L$ and $T_R$ are height-$n$ subtrees of $G$ rooted at entrance and exit, respectively. Furthermore, the vertices of $T_L$ and $T_R$ provide a bipartition of the vertices of $G$, and the edges of $T_L$, $T_R$, and WELD provide a tripartition of the edges of $G$.

We categorize the vertices of WELD depending on whether they belong to $T_L$ or $T_R$, and on the colors of the edges joining them to non-WELD vertices.

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

As an example, the vertices colored lavender and plum in Figure 3a 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 49). Note that Definition 47 partitions the vertices of WELD into 6 parts. The following definition is crucial for describing a distribution of welded tree graphs that is classically “hard”.

\begin{definition} [Color-preserving permutations]
A permutation $\sigma$ of the vertices of WELD is a color-preserving permutation if for any vertex $v$ of WELD, and any $c \in C$, $v$ is $c$-left (respectively $c$-right) iff $\sigma(v)$ is $c$-left (respectively $c$-right). We will sometimes refer to a color-preserving permutation $\sigma$ as a permutation of $V_D$ that acts as $\sigma$ on vertices of WELD.
\end{definition}
Figure 3 Example of a color-preserving permutation $\sigma$. The permutation $\sigma$ is the identity permutation except that it maps the vertex colored lavender to the vertex colored plum. Note that the resulting graph $G^\sigma$ is a valid 3-colored welded tree graph.

and as identity on non-WELD vertices. For any color-preserving permutation $\sigma$, let $\text{WELD}^\sigma$ denote the graph obtained by applying $\sigma$ to the vertices of WELD. For any color-preserving permutation $\sigma$, let $G^\sigma$ denote the graph on vertex set $V_G$ obtained by permuting the WELD edges of $G$ according to $\sigma$ (and leaving the rest of the graph $G$ as it is): if $u, v$ are WELD vertices, then there is an edge of color $c \in C$ in $G$ joining vertices $u$ and $v$ if and only if there is an edge of color $c$ joining vertices $\sigma(u)$ and $\sigma(v)$; otherwise, there is an edge of color $c \in C$ in $G^\sigma$ joining vertices $u$ and $v$ if and only if there is an edge of color $c$ in $G^\sigma$ joining vertices $\sigma(u)$ and $\sigma(v)$.

Figure 3 illustrates an example of a color-preserving permutation. Note that $V_{G^\sigma} = V_G$, and the non-WELD vertices and edges of $G$ remain invariant under $\sigma$. Therefore, the induced subgraph of $G$ on vertices of $T_L$ (respectively $T_R$) is $T_L$ (respectively $T_R$). Moreover, $\text{WELD}^\sigma$ is the induced subgraph of $G^\sigma$ on vertices of WELD. One can verify that the graphs obtained by applying color-preserving permutations on $G$ are valid 3-colored welded tree graphs.

**Lemma 49.** Let $\sigma$ be a color-preserving permutation. Then $G^\sigma$ is a valid 3-colored welded tree graph.

Recall that Definition 2 specifies the classical oracle function $\eta^c_\sigma$ for each $c \in C$ associated with $G^\sigma$ for the identity permutation $\sigma$. The following definition generalizes this by specifying the classical oracle function $\eta^c_\sigma$ for each $c \in C$ associated with $G^\sigma$ for any color-preserving permutation $\sigma$.

**Definition 50.** Let $V_G$, $I_c$, and $N_c$ for each $c \in C$, NOEDGE, and INVALID be defined as in Definitions 1 and 2. Let $V_{\text{WELD}}$ refers to the set of vertices of WELD. For any color-preserving permutation $\sigma$, let $\eta^c_{\text{WELD}}(v)$ be $\sigma(\eta_c(\sigma^{-1}(v)))$ whenever $v, \eta_c(v) \in V_{\text{WELD}}$ and $\eta_c(v)$ otherwise. Let $\eta^c := \{\eta^c_\sigma : c \in C\}$ be the oracle corresponding to the color-preserving permutation $\sigma$. 

We now define the notion of path-embedding in $G^\sigma$ 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 51 (Path-embedding).** Let $\sigma$ be any color-preserving permutation. Let $\ell \in [p(n)]$ and $t \in C^\ell$. That is, $t = (c_1, \ldots, c_\ell)$ for some $c_1, \ldots, c_\ell \in C$. Then, define the path-embedding of $t$ under the oracle $\eta^\sigma$, denoted by $\eta^\sigma(t)$, to be a length-$\ell$ tuple of vertex labels as follows. The first element of $\eta^\sigma(t)$ is $\eta^\sigma(\text{Entrance})$ whereas for $j > 1$, the $j$th element of $\eta^\sigma(t)$ is $\eta^\sigma_j(\eta^\sigma(t)_{j-1})$. 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 that $\eta^\sigma(t)$ 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 $G^\sigma$ if it encounters a sequence of vertices and edges that forms a cycle in $G^\sigma$.

We restrict our attention to the color sequences that do 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 describe notation for each time a certain path-embedding crosses the WELD so that we can refer to the tree that it goes to and the WELD edge that it goes through.

**Definition 52.** Let $\sigma$ be any color-preserving permutation and let $t$ be any sequence of colors that does not contain even-length palindromes. We use $T^\sigma_i$ to denote the $i$th subtree and $e^\sigma_i$ to denote the $i$th edge of the WELD encountered by the path-embedding $\eta^\sigma(t)$. Furthermore, let $\ell^\sigma(t)$ denote the number of subtrees encountered by the path-embedding $\eta^\sigma(t)$.

Note that the number of WELD edges encountered by the path-embedding $\eta^\sigma(t)$ is $\ell^\sigma(t) - 1$. For each $i \in [\ell^\sigma(t) - 1]$, the edge $e^\sigma_i$ joins a vertex in $T^\sigma_i$ to a vertex in $T^\sigma_{i+1}$.

The following definition formalizes statements (i) and (ii) that we described intuitively at the beginning of this section.

**Definition 53.** Let $\sigma$ be any color-preserving permutation and let $t$ be any sequence of colors that does not contain an even-length palindrome. We say that $t$ has small displacement if after encountering a WELD vertex, 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. We say that $t$ is non-colliding if for any edge $e$ joining some leaf of $T^\sigma_{i+1}$ with some leaf of $T^\sigma_i$ for some $i, j \in [\ell^\sigma(t)]$, $e$ must be $e^\sigma_i$ or $e^\sigma_j$. We say that $t$ is desirable if $t$ has small displacement and is non-colliding.

Note that, in the above definition, if $e = e^\sigma_i$, then $j = i + 1$ and if $e = e^\sigma_j$, then $j = i - 1$. Thus, $t$ being non-colliding essentially means that if there is an edge $e$ between trees $T^\sigma_i$ and $T^\sigma_j$ for some $i, j \in [\ell^\sigma(t)]$, then $j = i + 1$ or $j = i - 1$, and $T^\sigma_i$ and $T^\sigma_j$ are not joined by any edge other than $e$.

It is easy to see that for any sequence of colors $t$ that does not contain an even-length palindrome, beginning from the entrance and following the sequence of colors specified by $t$ will not result in reaching the exit if $t$ has small displacement, and will not result in going through a cycle if $t$ is non-colliding. The following lemma is crucial for our argument in this section, which essentially argues that for a fixed sequence of colors and a uniformly random color-preserving permutation $\sigma$, it is improbable for the corresponding path-embedding to contain the exit or a path that forms a cycle in $G^\sigma$.

**Lemma 54.** Let $\ell \in [p(n)]$ and $t \in C^\ell$ such that $t$ does not contain an even-length palindrome. Choose the permutation $\sigma$ according to the distribution $D_n$. Then the probability that the path-embedding $\eta^\sigma(t)$ encounters the exit or a cycle in $G^\sigma$ is at most $4p(n)^2 \cdot 2^{-n/3}$. 


We can extend the result of Lemma 54 about polynomial-length sequences of colors to polynomial-size subtrees of the address tree \( T \) (see Definition 11). For this purpose, we define the notion of subtree-embedding of subtrees of \( T \).

**Definition 55 (Subtree-embedding).** Let \( \sigma \) be any color-preserving permutation. Let \( \ell \in [p(n)] \) and \( t \in \mathbb{C}^{\ell} \). Let \( T \) be a subtree of the address tree \( T \) of size \( p(n) \) that contains the vertex labeled EMPTYADDRESS but does not contain vertices having labels in \( \text{SpecialAddresses} \setminus \{\text{EMPTYADDRESS}\} \). For any vertex of \( T \) labeled by \( t \neq \text{EMPTYADDRESS} \), let \( c_{(t)} \) denote the last color appearing in the sequence \( t \) and let \( \text{pre}(t) \) denote the color sequence formed by removing the last color from \( t \). Define the subtree-embedding of \( T \) under the oracle \( \eta^\sigma \), denoted \( \eta^\sigma(T) \), to be a tree isomorphic to \( T \) whose vertex labels are in \( V_G \) and specified as follows. The vertex \( \eta^\sigma(T)_t \) of \( \eta^\sigma(T) \) corresponding to the vertex of \( T \) labeled by \( t \) is \( \text{ENTRANCE} \) if \( t = \text{EMPTYADDRESS} \) and is \( \eta^\sigma_{\text{pre}(t)}(\eta^\sigma(T)_{\text{pre}(t)}) \) otherwise. We say that the subtree-embedding \( \eta^\sigma(T) \) encounters the \( \text{EXIT} \) if it contains a vertex labeled \( \text{EXIT} \) and that \( \eta^\sigma(T) \) encounters a cycle if it contains two vertices having the same label.

For any tree \( T \) specified in Definition 55, the root of \( T \) will always be \( \text{EMPTYADDRESS} \), so the root of \( \eta^\sigma(T) \) will always be \( \text{ENTRANCE} \). The subtree-embedding \( \eta^\sigma(T) \) of a tree \( T \) will correspond to the subgraph of \( G^\sigma \) that contain vertices which can be reached by following the addresses given by vertex labels of \( T \) in \( G^\sigma \) beginning at the \( \text{ENTRANCE} \).

Next, we show that for a fixed sub-tree of \( T \) and a randomly chosen color-preserving permutation \( \sigma \), it is not possible for the corresponding subtree-embedding to contain the \( \text{EXIT} \) or a path that forms a cycle in \( G^\sigma \), except with exponentially small probability.

**Lemma 56.** Let \( T \) be a subtree of the address tree \( T \) of size \( p(n) \) that contains the vertex labeled EMPTYADDRESS but does not contain vertices having labels in \( \text{SpecialAddresses} \setminus \{\text{EMPTYADDRESS}\} \). Let the permutation \( \sigma \) be chosen according to the distribution \( D_n \). Then the probability that the subtree-embedding \( \eta^\sigma(T) \) encounters the \( \text{EXIT} \) or a cycle is at most \( 4p(n)^{42}2^{-n/3} \).

We conclude this section 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 \( \text{EXIT} \) or a cycle in the welded tree graph sampled according to this distribution.

**Theorem 57.** Let \( S \) be the set of subtrees of the address tree \( T \) of size \( p(n) \) that contain the vertex labeled EMPTYADDRESS but do not contain vertices having labels in \( \text{SpecialAddresses} \setminus \{\text{EMPTYADDRESS}\} \). Then there exists a distribution \( D_n \) over size-\( n \) 3-colored welded tree graphs \( G^\sigma \) such that for any classical query algorithm \( A_{\text{classical}} \) that samples a tree \( T \) from \( S \) and computes the associated subtree-embedding in \( G^\sigma \), the probability that \( A_{\text{classical}} \) finds the \( \text{EXIT} \) or a cycle in \( G^\sigma \) is at most \( 4p(n)^{42}2^{-n/3} \).

**References**


