Time-Space Trade-offs in Searching for a Path in Welded Trees: Classical vs. Quantum Approaches

Yvan Le Borgne[1] and Shrinidhi Teganahally Sridhara[1*]

[1]Univ. Bordeaux,CNRS, Bordeaux INP, LaBRI, UMR 5800, [F-33400 Talence], France.

*Corresponding author(s). E-mail(s): shrinidhi.teganahally-sridhara@u-bordeaux.fr;
Contributing authors: borgne@labri.fr;

Abstract

Welded Trees are a class of graphs designed to exhibit exponential algorithmic speedup for certain search problems via quantum walks, outperforming any classical algorithm. A Welded Tree of depth d consists of two complete binary trees, each of depth d, whose leaves are interconnected by an additional cycle or permutation alternating between the two trees. Access to this graph is provided only through queries to an oracle, which, given a vertex, returns its neighbors. In 2021, Aaronson suggested the Finding Path To Exit problem, which consists of finding a *path* to the root of the opposite tree given the root of one tree. This can be seen as an extension of the Finding Exit problem, where the task is simply to find the root of the opposite tree, for which Childs et al. demonstrated an exponential speed-up in 2003.

We provide a quantum algorithm for solving the Finding Path To Exit problem that performs $\tilde{O}(2^{d/3})$ queries and also formalize the folklore lower bound that any classical algorithm requires at least $\tilde{\Omega}(2^{d/2})$ queries. This is the first proof of a *polynomial speed-up* for this problem over any classical algorithm. In addition, we present and analyze several other algorithms, focusing on the trade-offs between queries and classical or quantum space.

Keywords: Discrete quantum walks, Search in graphs, Quantum speed-up

1 Introduction

Since the advent of the theory of quantum computation, there has been a quest to find functions that are provably faster to compute using quantum computation compared to the traditional Turing machine based classical computation. Famous examples include the Shor's Algorithm for factoring and the discrete logarithm problem [1], Grover's algorithm for unstructured search [2] and others. Almost all the observed speedups are in terms of query complexity and measure the number of times these algorithms query an oracle in order to find a solution with high probability. In the realm of graph problems, quantum computing has been proven to offer significant speed-ups compared to classical algorithms. Welded Trees are one such examples of graphs which were designed to exhibit an exponential algorithmic speed-up using quantum walks over any classical algorithm by Childs, Cleve, Deotto, Farhi, Guttman and Spielman in 2003 [3]. They have been extensively studied in Quantum Walks [4–6], Quantum Adiabatic Computation [7] and Graph Property testing [8].

Welded Trees are family of graphs G_d, obtained by joining the leaves of two complete binary trees of depth d using two random perfect matchings or a random cycle(see Figure 1). For simplicity, we may assume that they come along with an additional proper edge-colouring to be described later. Access to these graphs are provided only via an oracle $\mathcal{O}_W$. This oracle $\mathcal{O}_W$, when given a pair (v, c) as input, where v is a vertex in the welded tree and c is a colour, outputs the function $f_c(v) = u$, where u is the vertex that is connected by an edge of color c to the vertex v or 0 if no such edge exists. In terms of quantum computation, the oracle is defined as the following unitary operator,

$$\mathcal{O}_W \left|v, c, x\right\rangle = \left|v, c, x \oplus f_c(v)\right\rangle .$$

Generally speaking, we are provided with an adjacency list oracle, where each query reveals only the neighboring nodes, thus granting us access to only a local view of the graph.

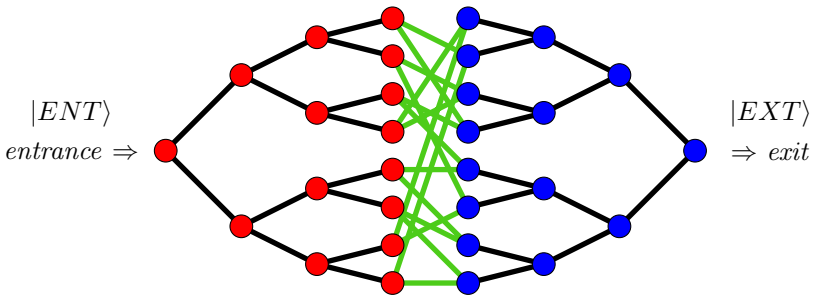

Fig. 1: A sample of welded trees of depth $d = 3$

We denote the roots of the two binary trees that form a welded tree as the *entrance* and the *exit*. The set of vertices at distance k from *entrance* form the vertices of

column k. We denote them by C_k. Hence, *exit* is the only vertex in column $(2d+1)$. The degree of every vertex in a welded tree is three, except for *entrance* and *exit*.

One of the natural search problems on these welded trees is the task of finding *exit* vertex, given the *entrance* vertex and oracle. Moreover, the vertices of a welded tree of depth d are labelled by bit strings of length $2d$, chosen randomly and independently where d is the depth of the welded tree. We refer to this bit string that represents a vertex v as the *id* of that vertex. The map from vertices to these *id*s as label where $\mathsf{label} : V \to \{0,1\}^{2d}$ (see a more detailed example in Figure 4). Below is a formal definition of the *exit* search problem.

Definition 1 FINDING EXIT: Given the *id* of the *entrance* of a welded tree of depth d and access to the adjacency oracle $\mathcal{O}_W$, return the *id* of the *exit*.

In 2003, Childs et al. [3] designed a quantum algorithm that efficiently solves the FINDING EXIT problem in $\mathcal{O}(poly(d))$ many queries to the oracle. They also proved that in contrast, *any* classical randomised algorithm would have to make $\Omega(2^{d/6})$ many queries to the oracle to return *exit* with high probability and thus demonstrated exponential algorithmic speed-up using quantum computation in the query model. The quantum algorithm for FINDING EXIT in [3] is based on quantum walks and notably focuses only on the current vertex, without maintaining a partial path from the *entrance*, which allows for destructive interferences that are crucial for achieving a speed-up in the number of queries. The lower bound for a randomised classical algorithm was improved by Fenner and Zhang [9] to $\Omega(2^{d/3})$ queries. In folklore, the lower bound is assumed to be $\Omega(2^{d/2})$. Though the original quantum algorithm was based on continuous quantum walks, subsequent works provide discrete quantum walk and multidimensional quantum walk based algorithms with improved query complexity of $\mathcal{O}(d^2 \log(d))$ and $\mathcal{O}(d)$ respectively.

In the classical scenario, any algorithm capable of identifying the *exit* can be adapted, with additional memory, to trace a **path** from the *entrance* to the *exit* in a welded tree. Surprisingly, while we can efficiently find the *exit* vertex quantumly, constructing a path from the *entrance* to the *exit* with a provably lesser number of queries than classical computation remains unresolved. Aaronson [10], in 2021, defined this natural extension of the FINDING EXIT problem as FINDING PATH TO EXIT and highlighted that the question of whether FINDING PATH TO EXIT can be solved using a quantum algorithm using $2^{o(d)}$ queries as one of the challenging open problems in quantum query complexity.

Definition 2 FINDING PATH TO EXIT: Given the *id* of the *entrance* of a welded tree of depth d and access to the adjacency oracle $\mathcal{O}_W$, find a **path** of vertices $v_0 v_1 \dots v_n$ where v_0 is the *entrance* vertex and v_n is the *exit* vertex of the given welded tree..

Solving FINDING PATH TO EXIT in the quantum realm is widely acknowledged to be considerably more challenging than FINDING EXIT. This difficulty is highlighted by a recent 2022 exponential lower bound established by Childs et al. [11] for a specific

family of quantum algorithms that maintain complete path information from the entrance.

Our Results

In this paper, we look at the Finding Path To Exit from the perspectives of both classical and quantum algorithms and analyse the query and space complexity trade-offs. Our primary contribution is a quantum algorithm for the Finding Path To Exit with a query complexity of $\tilde{\mathcal{O}}(2^{d/3})$[1]. Classically, there is an algorithm that solves this problem using $\tilde{\mathcal{O}}(2^{d/2})$ due to Shalev Ben-David [12]. This is optimal for any classical algorithm as it matches the folklore lower bound of $\Omega(2^{d/2})$ on the exponential order which we prove formally in subsection 5.2. Thus our $\tilde{\mathcal{O}}(2^{d/3})$ algorithm is, to the best of our knowledge, the first quantum algorithm for Finding Path To Exit that provably outperforms classical methods in finding a path to the exit by providing a polynomial speedup over the classical counterpart. This algorithm utilises classical memory of $\tilde{\mathcal{O}}(2^{d/3})$ and quantum memory of $\mathcal{O}(poly(d))$ qubits. Below is the statement of the main result of this paper.

Theorem 1 *There is a polynomial speed-up, from $\tilde{\Theta}(2^{d/2})$ queries classicaly to $\tilde{\mathcal{O}}(2^{d/3})$ quantumly, for Finding Path To Exit problem in a welded tree of depth d.*

On the other hand we also introduce another quantum algorithm for the same problem with a complexity of $\mathcal{O}(2^{d/2})$ queries, but with a space complexity of $\mathcal{O}(d^2)$. It is worth to note that both of our quantum algorithms always find a *shortest* path from *entrance* to *exit* and differ slightly from the restricted family of algorithms previously proposed by Childs et al. [11] for establishing the lower bound. We also look at two different types of classical random walk based algorithms for Finding Path To Exit namely the backtracking random walk and non-backtracking random walk with exponential number of queries but polynomial space. The analysis of the expected behaviours of these two random walks on welded trees is based on explicit generating functions. We prove of the folklore lower bound of $\Omega(2^{d/2})$ queries for classical algorithms solving Finding Path To Exit by considering the difficulty of finding cycles. Here is an informal statement of the lower bound theorem.

Theorem 2 *Any classical randomized algorithm that makes $\mathcal{O}(2^{\alpha d})$ queries to the oracle, where $\alpha \in [0, 1/2)$, cannot solve the Finding Path To Exit problem with probability greater than $\tilde{\mathcal{O}}(2^{(2\alpha-1)d})$.*

In Figure 2, we compare complexity of several classical and quantum algorithms, including ours, for the Finding Path To Exit problem on welded trees.[2]

[1] Henceforth tilde hides logarithmic and polynomial factors in d

[2] Here, time is defined as the number of queries to the oracle, and space refers to the number of bits or qubits required. Typically, these measurements do not account for the space needed to implement the classical or quantum version of the oracle.

Algorithm	Time (Average)	Space (Average)	References
1. Best classical algorithm	$\tilde{\mathcal{O}}(2^{d/2})$	$\tilde{\mathcal{O}}(2^{d/2})$ bits	[12]+ subsubsection 5.1.2
2. Its lower memory version with $\gamma \leq \alpha \leq 1$	$\mathcal{O}(2^{(1-\alpha)d+(\alpha-\gamma)d(d+1)})$	$\tilde{\mathcal{O}}(2^{\gamma d})$ bits	subsubsection 5.1.3
3. Our 1^{st} quantum algorithm	$\tilde{\mathcal{O}}(2^{d/2})$	$\tilde{\mathcal{O}}(2^{d/2})$ bits, $\mathcal{O}(d)$ qubits	subsection 3.1 + [4]
4. Its lower memory version	$\tilde{\mathcal{O}}(2^{d/2})$	$\tilde{\mathcal{O}}(d^2)$ bits, $\mathcal{O}(d)$ qubits	subsection 3.3 +[13]
5. Random walk	$\left(\frac{4\times 2^d-3}{2^{d+1}}\right)\cdot 2\cdot(3\cdot 2^d-2)$	$2d$ bits	[14]+ section 4
6. Non-backtracking random walk	$\left(\frac{2\times 2^d-1}{2^{d+1}}\right)\cdot 2\cdot(3\cdot 2^d-2)$	$2d+\lceil \ln_2 C\rceil$ bits	section 4
7. Our 2^{nd} quantum algorithm	$\tilde{\mathcal{O}}(2^{d/3})$	$\mathcal{O}(d^2)$ qubits $\mathcal{O}(2^{d/3})$ bits	subsection 3.2
1. Classical lower bound	$\tilde{\Omega}(2^{d/2})$		subsection 5.2, [3, 9]
2. Quantum lower bound	$\Omega(exp(d))$ (with restrictions)		[11]

Fig. 2: Time and space complexity of several algorithms for FINDING PATH TO EXIT problem.

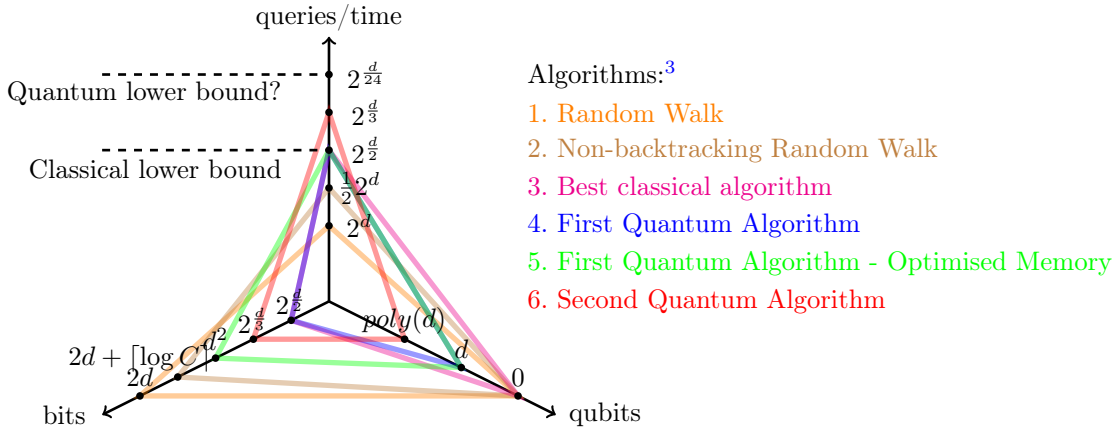

Fig. 3: Different algorithms and their query vs space comparisions.

Our methods

Both our quantum algorithms utilize efficient quantum algorithm for FINDING EXIT as a quantum primitive to identify the *exit* vertex. Then we convert the problem of finding a path into a collision problem involving the task of finding two partial

non-backtracking paths from *entrance* and *exit* to vertices in column $d+1$. Based on the way we solve this collision problem, we get two different algorithms with different complexities. If we decide to solve it classically, it involves the sampling of non-backtracking walks of lengths $d+1$ from the *entrance* or d from the exit, and awaiting a collision among vertices in column $d+1$ at the conclusion of these paths. Leveraging a variant of the Birthday paradox guarantees an average query count of $\tilde{\mathcal{O}}(2^{d/2})$, with substantial classical memory $\tilde{\mathcal{O}}(2^{d/2})$ to store the sampled paths. Standard meet-in-the-middle algorithms, as discussed in [13], enable maintaining the time complexity at $\tilde{\mathcal{O}}(2^{d/2})$ while limiting the memory to $\mathcal{O}(d^2)$ bits and qubits.

Our second quantum algorithm improves the collision resolution time to $\tilde{\mathcal{O}}(2^{d/3})$ using the standard quantum search techniques of amplitude amplification [15] and Grover's algorithm [2]. Initially we sample some non-backtracking paths from *exit* to column $d+1$. Then, we build a quantum state that is a superposition of non-backtracking paths from *entrance* of length $d+1$ and later try to find a *good* path or in other words, a path which has the same last vertex as one of the classically sampled paths from the *exit*. Since we can efficiently check if a path is *good* or not, we can make use of this to attain a polynomial speedup.

In a classical random walk algorithm on welded tree, the walker uses $2d$ bits to store the *id* of current vertex and samples neighbors randomly, reflecting a significant memory efficiency. The average number of oracle queries is precisely evaluated using a formula based on commuting time by Tetali [14]. Recording the last edge color during the random walk adds $\lceil \ln_2(C) \rceil$ bits to memory but allows the random walk to become non-backtracking, thus avoiding revisiting vertices. But surprisingly, this just reduces the query complexity only by a factor asymptotically close to 2 despite using a bit of extra information. This can be attributed to the fact that once we reach the welded region, we lose the information of whether we are moving towards or away from the *exit*. The average number of queries to the oracle is accurately evaluated using a formula involving edge count and a resistance-like term.

Organisation of the rest of the paper

section 2 introduces some preliminary definitions and outlines the sampling of welded trees and their oracle. In section 3 we present three quantum algorithms including the one with query complexity of $\tilde{\mathcal{O}}(2^{d/3})$ leading to the proof of Theorem 1. section 4 gives exact analyses of the average number of queries for two classical algorithm with a very small memory namely random walks and non-backtracking random walks. subsection 5.2 contains the proof of Theorem 2. We summarise our results and list some open questions in section 6.

2 Preliminaries

In a welded tree W of depth d, we refer to C_x, the set of vertices at distance x from the *entrance*, as *column* x. For example in Figure 1, the welding permutation alternates between vertices in columns d and $d+1$. A walk of n steps in a welded tree is a sequence of vertices $w = w_0, w_1 \cdots w_n$ with w_i and w_{i+1} connected by an edge. A walk is *backtracking* if there exists an index i such that $w_i = w_{i+2} \neq w_{i+1}$. If not, it is called

as a *non-backtracking* walk. We denote the last vertex of a walk $w = (w_i)_{i=0,\dots\ell(w)}$ by $\mathsf{last}(w)$. For a set C of walks, $\mathsf{last}(C) := \{\mathsf{last}(w) | w \in C\}$ is the set of last vertices of these walks. If $\mathsf{last}(w) = \mathsf{w}'$ for any two walks w and w', then we say that there is a *collision* between these two walks. The following fact about non-backtracking walks on welded trees can be easily verified.

Proposition 3 *For every non-backtracking walk $w = w_0 w_1 \cdots w_{d+1}$ rooted at* entrance*, vertex w_i belongs to C_i where $i \leq d+1$. Analogously, for every non-backtracking walk $w = w_0 w_1 \cdots w_{d+1}$ rooted at* exit*, the vertex w_i belongs to C_{2d+1-i}, where $i \leq d+1$.*

In fact, we can sample a non-backtracking walk uniformly at random from roots of welded tree by considering the two possible vertices at each step with equal probability.

The choice of an instance of welded trees and their oracle is already described in seminal work on FINDING EXIT problem [3] but we provide here a more explicit version later used in our proofs. The aim of oracle description for welded trees is to hide a part of its global structure, preserving a coherent local view. We detail here an explicit random sampling of such an oracle that will be used in particular in the probabilistic analysis of classical lower bounds. For a pair of welded trees of depth d we sample independantly and uniformly among the possible elements the following data $\mathcal{D} = ((\sigma^{[x,x+1]})_{x=0,\dots 2d}, \mathsf{label}, (\tau_v)_{v\in V})$ where:

- the edges between column x and $x+1$ are encoded by permutations $(\sigma^{[x,x+1]})_{x=0,\dots 2d}$ where $\sigma^{[x,x+1]}$ is a random permutation of size $2^{\min(x+1,2d+1-x)}$,
- the relabeling map $\mathsf{label} : V := \{0,\dots 2(2^{d+1}-1)-1\} \to \{1,\dots 2^{2d}-1\}$ is a random injective function whose co-domain avoids 0 and for each vertex v the oddity of $\mathsf{label}(v)$ is the oddity of the distance of v to the *entrance* vertex which is the root of left welded tree,
- the half-proper coloring $(\tau_v)_{v\in V}$ is made up of random permutations of size 3 indexed by vertices of welded trees.

We consider an edge $e = (u, v)$ of the welded trees where w.l.o.g u is at even distance d from entrance vertex and v at odd $d \pm 1$ distance. This edge e is decomposed into two half-edges (u, e) and (v, e) respectively (half-colored) by $\tau_u((u, e))$ and $\tau((v, e))$ where we use a fixed ordering of half-edges incidents to u, respectively v. The color c of the edge e is the pair $(\tau_u((u, e)), \tau_v((v, e)))$. In this case, the classical oracle $f_c(\cdot)$ is defined by

$$f_{(\tau_u((u,e)),\tau_v((v,e)))}(\mathsf{label}(u)) := \mathsf{label}(v) \text{ and } f_{(\tau_u((u,e)),\tau_v((v,e)))}(\mathsf{label}(v)) := \mathsf{label}(u)$$

if $\sigma^{[d,d\pm1]}((u, e)) = (v, e)$ using a fixed ordering (from top to bottom here) of half-edges in each direction between column C_d and $C_{d\pm1}$ where $\sigma^{[d,d-1]} = \left(\sigma^{[d-1,d]}\right)^{-1}$, vertices are replaced by their label and otherwise the oracle returns 0. For example, on Figure 4, $f_{(0,2)}(95) = 44$, $f_{(1,1)}(50) = 63$, $f_{(0,1)}(56) = 0$, $f_{(1,2)}(52) = 63$, ...

The label targets a sufficiently large codomain such that the probability for any element to have a pre-image is very small, lower than 2^{-d+2}. Hence any algorithm sampling random label needs at least an average of 2^{d-2} calls to the oracle to find

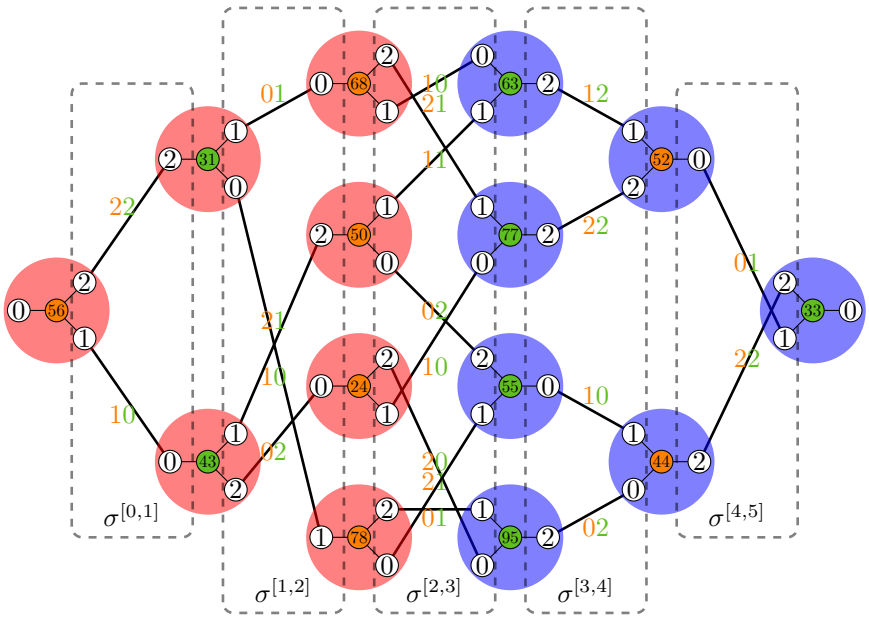

Fig. 4: Oracle for a pair of welded trees of size $d = 2$ deduced from sampled data $\mathcal{D} = (\sigma^{[0,1]} = 01, \sigma^{[1,2]} = 0312, \sigma^{[2,3]} = 20147365, \sigma^{[3,4]} = 0123, \sigma^{[4,5]} = 10, \dots)$

a label of vertex of welded trees. Since all our algorithms will have at most $\mathcal{O}(2^{d/2})$ in average calls to the oracle, none of them will sample possible labels, hence only discovers new labels of vertices via calls to the oracle. This was a natural assumption on algorithms recently made by Childs & al.[11].

Notice that all information about the welded trees are hidden by randomly sampled elements $\sigma^{[x,y]}$, τ_v, label of $\mathcal{D}$ expect the color green or orange of the current vertex v that is known from the oddity of the number of steps from the *entrance* in this bipartite graph and the fact that we consider only paths extended by oracle from the *entrance* (or exit) of defined oddity of distance to entrance.

3 Quantum algorithms (for Finding Path To Exit problem)

In this section we present three quantum algorithms for Finding Path To Exit problem. All these algorithms begin by finding the *exit* by solving Finding Exit problem using efficient quantum algorithms. We observe that what remains is indeed a collision problem. The path-finding problem is reduced to a collision problem by splitting the full path into two partial non-backtracking paths—one from the *entrance* and one from the *exit* and searching for a pair that have a collision (see section 2). Finding such a collision effectively reconstructs a complete path from *entrance* to *exit*. In subsection 3.1 we present the first algorithm which solves this collision problem

classicaly, using $\tilde{\mathcal{O}}(2^{d/2})$ queries and $\mathcal{O}(2^{d/2})$ space[4]. In subsection 3.2, we optimize the number of queries to obtain the second algorithm that solves the collision problem quantumly in $\tilde{\mathcal{O}}(2^{d/3})$ queries. In subsection 3.3, we optimize the memory classicaly to obtain the third algorithm performing $\tilde{\mathcal{O}}(2^{d/2})$ queries and using $\mathcal{O}(d^2)$ space.

3.1 Solving the collision problem classically

Our first Algorithm 1 to find a path to the *exit* is decomposed in two steps. The first step is to find the *exit* vertex using $\mathcal{O}(d)$ queries to the oracle using the current best quantum algorithm of Jeffery and Zur [4]. The second step involves searching classically for a path between *entrance* and *exit* by sampling non-backtracking walks of length $d+1$ and d from *entrance* and *exit* respectively, hoping for a collision between their end vertices.

Algorithm 1 Using classical collision algorithm to find a path from *entrance* to *exit*

Input: Welded tree W of depth d, oracle $\mathcal{O}_W$ and the *id* of the *entrance*
Output: A shortest path from *entrance* to *exit* of the welded tree W.

procedure FirstQuantumAlgorithm(*entrance*, $\mathcal{O}_W$)
  exit $\leftarrow$ Find *exit* of W using $\mathcal{O}(d)$ query algorithm of Jeffery and Zur [4].
  $A \leftarrow \emptyset$ ▷ Stores non-backtracking walks of length $d+1$ from *entrance*.
  $B \leftarrow \emptyset$ ▷ Stores non-backtracking walks of length ds from *exit*.
  while $\mathsf{last}(A) \cap \mathsf{last}(B) = \emptyset$ **do**
    $X \leftarrow$ A non-backtracking walk of length $d+1$ sampled randomly from *entrance*.
    $A \leftarrow A \cup \{X\}$
    $Y \leftarrow$ A non-backtracking walk of length d sampled randomly from *exit*.
    $B \leftarrow B \cup \{Y\}$
  end while
  $v \leftarrow \mathsf{last}(A) \cap \mathsf{last}(B)$
  $w_A \leftarrow$ A walk in A with last vertex v
  $w_B \leftarrow$ A walk in B with last vertex v
  $w_B \leftarrow rev(w_B)$ ▷ Vertices of w_B reversed.
  return $w_A \cdot w_B$
end procedure

Lemma 1 *Given a welded tree W of depth d,* FirstQuantumAlgorithm *finds a shortest path between* entrance *and* exit *with probability at least $\frac{1}{4}$ in $\mathcal{O}(d \cdot 2^{d/2})$ queries to the oracle $\mathcal{O}_W$.*

Proof The correctness of this algorithm relies on Proposition 3. Thus $\mathsf{last}(A)$ and $\mathsf{last}(B)$ contain vertices in $C_{d+1} = C_{2d+1-d}$. Once $\mathsf{last}(A) \cap \mathsf{last}(B)$ is non-empty, it implies that there

[4]By space, we mean both classical and quantum memory.

exists walks $w_A \in A$ and $w_B \in B$, the concatenation of these two results in a walk of length $2d+1$ between the *entrance* and the *exit* which happens to be the shortest path as well.

The oracle design in Section 2 ensures that the final vertices in $\mathsf{last}(A)$ and $\mathsf{last}(B)$ are chosen uniformly at random from the 2^d vertices in C_{d+1}, with the permutations $\sigma^{[d,d+1]}$ and $\left(\sigma^{[d+1,d+2]}\right)^{-1}$. This creates a variant of the Birthday paradox known as Claw finding, where the condition $\mathsf{last}(A) \cap \mathsf{last}(B)$ involves a codomain of size $r = 2^d$, as discussed in II.8 on page 116 of [16]. Let $\mathbf{B}$ be the random variable representing the number of uniform samples needed to obtain two identical elements from a set of size r. According to [16], $\Pr(\mathbf{B} > t\sqrt{r}) \sim e^{-t^2/2}$. For $t = \sqrt{2\log 2}$, this gives $\Pr(\mathbf{B} > \sqrt{2r\log 2}) \sim \frac{1}{2}$, so $\Pr(\mathbf{B} \leq \sqrt{2r\log 2}) \sim \frac{1}{2}$. Thus, after $\frac{\sqrt{2r\log 2}}{2} = \sqrt{\frac{\log 2}{2}}2^{d/2}$ iterations of the while loop, there is a 50% chance of a collision, which may correspond to the concatenation of paths from the *entrance* (in $\mathsf{last}(A)$) and the *exit* (in $\mathsf{last}(B)$). Therefore, after $\sqrt{\frac{\log 2}{2}}2^{d/2}$ iterations, the success probability is at least 50%. Each iteration requires $\mathcal{O}(d)$ oracle calls, so Algorithm 1 finds a shortest path between the *entrance* and *exit* vertices on welded trees with a success probability of at least $\frac{1}{4}$ while making at most $\mathcal{O}(d \cdot 2^{d/2})$ queries to $\mathcal{O}_W$. □

3.2 A $\tilde{\mathcal{O}}(2^{d/3})$ quantum algorithm for Finding Path To Exit

In this section we show that using standard quantum techniques, we can solve this collision problem better, leading to an algorithm that finds a shortest path from *entrance* to *exit* in $\mathcal{O}(2^{d/3})$ many queries to the oracle $\mathcal{O}_W$.

We first observe that given a bit string $s \in \{0,1\}^{d+1}$, we can uniquely map it to a vertex $v(s)$ in column $d+1$. This is because, since a welded tree is made of two binary trees and we can think of one of the two outgoing edges at every vertex is labelled 0 and the other is labelled 1. Now, based on the bit string s, we can traverse a non-backtraking path of length $d+1$ starting from *entrance* which leads us to a unique vertex in column $d+1$. But since the vertices are labelled randomly using bit strings from $\{0,1\}^{2d}$, we won't know the *id*s of the vertices along the path leading to the unique vertex $v(s)$ unless we query the oracle. Hence, we make use of the oracle $\mathcal{O}_W$ to obtain the *id*s of all vertices along this path for all possible $v(s)$ or equivalently all possible bit strings s.

In the previous section, we sampled equal number of non-backtracking paths from *entrance* and *exit*, of length $d+1$ and d respectively, and aimed to find a collision among the last vertices of these paths which are guaranteed to belong to column $d+1$. Given the fact that we can discover the path from bit strings, the task is to search for a **good** path from *entrance*, meaning a path whose last vertex is also the last vertex of a path in the sampled set of non-backtracking paths of length d from the *exit*. For this task we employ Grover's algorithm [2]. We would like to note that this identical to the idea behind the collision finding algorithm by Brassard, Høyer and Tapp [17].

In order to employ Grover, we need to have an oracle that distinguishes the **good** paths from others. We use T to denote the set of non-backtracking paths from *exit*

we have already sampled. Let $\mathcal{B}$ be the oracle such that,

$$|\psi\rangle := \mathcal{B}\,|0\rangle\,|0\rangle \to \frac{1}{\sqrt{2^n}} \sum_{x\in\{0,1\}^n} |x\rangle\,|P_x\rangle\,. \tag{1}$$

Let $\mathcal{O}_f$ be an oracle such that,

$$\mathcal{O}_f\,|x\rangle\,|P_x\rangle = (-1)^{f(x)}\,|x\rangle\,|P_x\rangle \tag{2}$$

$$\text{where, } f(x) = \begin{cases} 1, & \exists t \in T, \text{last}(P_x) = \text{last}(t) \\ 0, & \text{otherwise} \end{cases} \tag{3}$$

If such oracles exist, we could easily search for a path P_x that is marked by the function $f(x)$. And such a path when joined with the corresponding path from T, we get a shortest path from *entrance* to *exit*.

The following lemma shows that $\mathcal{B}$ can be constructed in $\tilde{\mathcal{O}}(d)$ many queries to the oracle $\mathcal{O}_W$.

Lemma 2 *If C be number of colors used in the proper edge-coloring of a welded tree, then with very high probability, the quantum state $|\psi\rangle$ in (1) can be constructed in atmost $\mathcal{O}(poly(\lceil \log C \rceil, d))$ many calls to the oracle $\mathcal{O}_W$.*

Proof We construct the state $|\psi\rangle$ as a coherent superposition over all non-backtracking paths from the entrance to the exit of the graph. Each vertex has three incident edge colors, exactly two of which correspond to valid non-backtracking steps. Since these two specific colors are unknown, a random query to the oracle $\mathcal{O}_W$ may yield a backtracking step (i.e., the previous vertex) or a null output. To resolve this, we use amplitude amplification to identify the two valid colors at each vertex.

Let $\mathcal{C}$ be the set of edge colors with $|\mathcal{C}| = C$, and define $t = |\{(c_1, c_2) \mid c_1 < c_2, c_1, c_2 \in \mathcal{C}\}|$. We begin by preparing the uniform superposition

$$|colors\rangle = \frac{1}{\sqrt{t}} \sum_{c_1<c_2} |c_1\rangle\,|c_2\rangle\,.$$

Given a vertex v_i (arrived at from v_{i-1}), there exist two valid non-backtracking colors $c_1^{v_i}$ and $c_2^{v_i}$. To identify them, we use two vertex registers $|a\rangle, |b\rangle$ (each of size $2d$), and three auxiliary qubits $|aux_1\rangle, |aux_2\rangle, |aux_3\rangle$ initialized to $|0\rangle$. We compute $\mathcal{O}_W\,|v_i, col_1, a\rangle$ and $\mathcal{O}_W\,|v_i, col_2, b\rangle$, and check whether the outputs correspond to valid forward steps (i.e., not equal to v_{i-1} and not the null vertex). If so, the respective auxiliary qubits are set to $|1\rangle$. If both are valid, we set $|aux_3\rangle = |1\rangle$. After uncomputing all intermediate registers except for $|aux_3\rangle$, the resulting state is

$$\frac{1}{\sqrt{t}}\,|v_{i-1}\rangle\,|v_i\rangle\,|c_1^{v_i}\rangle\,|c_2^{v_i}\rangle\,|0\rangle^{\otimes 4d+2}\,|1\rangle + \sqrt{\frac{t-1}{t}}\,|v_{i-1}\rangle\,|v_i\rangle \left(\sum_{(c_1,c_2)\neq(c_1^{v_i},c_2^{v_i})} |c_1\rangle\,|c_2\rangle \right) |0\rangle^{\otimes 4d+2}\,|0\rangle\,.$$

Using amplitude amplification [15], we boost the amplitude of the correct pair. Since $t \leq \binom{C}{2}$ is constant, only $\mathcal{O}(1)$ rounds are needed. This color identification process requires $\mathcal{O}(\text{poly}(d))$ queries to $\mathcal{O}_W$ and $\text{poly}(d, \lceil \log C \rceil)$ auxiliary qubits.

To construct $|\psi\rangle$, we begin with the initial state

$$|0\rangle^{d+1} |ENT\rangle \left(\bigotimes_{i=1}^{d+1} |0\rangle^{2d}\right) \left(\bigotimes_{i=1}^{d+1} |0\rangle^{2\lceil \log C \rceil}\right) |0\rangle^{4d+3},$$

where the first $d+1$ qubits represent a bit string $x \in \{0,1\}^{d+1}$, followed by the entrance vertex register, $d+1$ vertex registers for path construction, $d+1$ color registers, and $4d+3$ auxiliary qubits. We apply Hadamard gates to the first $d+1$ qubits to obtain a superposition over all bit strings x. Then, for each $i = 1$ to $d+1$, we apply the color identification procedure to obtain the valid colors $(c_1^{v_i}, c_2^{v_i})$ for v_i. Depending on the value of x_i, we apply $\mathcal{O}_W$ to extend the path by computing v_{i+1} from v_i using the selected color:

$$\text{if } x_i = 0: \quad \mathcal{O}_W |v_i, c_1^{v_i}, v_{i+1}\rangle, \qquad \text{else: } \mathcal{O}_W |v_i, c_2^{v_i}, v_{i+1}\rangle.$$

After $d+1$ such steps, the vertex registers encode the path $P_x = (v_1^x, v_2^x, \ldots, v_{d+1}^x)$ corresponding to each x, resulting in the desired state $|\psi\rangle$ (up to auxiliary qubits), which is a superposition over all non-backtracking paths from entrance to exit.

The entire procedure uses $\mathcal{O}(poly(d))$ queries to $\mathcal{O}_W$, requires $\mathcal{O}(\text{poly}(d, \log C))$ auxiliary qubits, and only $\mathcal{O}(1)$ rounds of amplitude amplification per vertex.

□

Lemma 3 *Given a welded tree W of depth d, there exists an algorithm that solves* Finding Path To Exit *on W in $\tilde{\mathcal{O}}(2^{d/3})$ queries to the oracle $\mathcal{O}_W$ and with a space complexity of $\tilde{\mathcal{O}}(2^{d/3})$ classical bits and $\mathcal{O}(d^2)$ qubits. Moreover the algorithm finds a shortest path from* entrance *to* exit.

Proof As a first step, we find the *exit* of the welded tree using $\mathcal{O}(d)$ queries. Now, we sample $2^{d/3}$ non-backtracking paths of length d from *exit*, which have their last vertices in column $d+1$. This involves $\mathcal{O}(d \cdot 2^{d/3})$ queries to the oracle $\mathcal{O}_W$. Using this classical information, we can build an oracle $\mathcal{O}_f$ where $\mathcal{O}_f |x\rangle = (-1)^{f(x)} |x\rangle |P_x\rangle$. A naive yet sufficient circuit of depth $\Theta(2^{d/3})$ for the computed oracle $\mathcal{O}_f$ consists in sequentially checking equality with each of the $\Theta(2^{d/3})$ binary words $z = z_1 \ldots z_{2d} \in \{\mathsf{last}(P) | P \in T\}$ via a (sub-)circuit $\bigotimes_{i=1}^{2d} X_i^{1-z_i}; \mathsf{C}^{2d}X_{aux}; \bigotimes_{i=1}^{2d} X_i^{1-z_i}$ involving a multi-CNOT gate targeting an auxiliary qubit that may then be turn into a phase via $\mathsf{H};\mathsf{X};\mathsf{H}$ followed by undoing the possible bitflip. It is very important to note that even though this oracle will have to look at each of the $2^{d/3}$ sampled paths to answer a query, it does not involve any query to the oracle $\mathcal{O}_W$. Hence this does not increase the query complexity of our algorithm.

Now we prepare the state $|\psi\rangle$ with very high probability using the steps described in the proof of Lemma 2. This state preparation requires only $poly(d)$ queries to $\mathcal{O}_W$.

Let $|G\rangle = \sum_{x:f(x)=1} |x\rangle |P_x\rangle$ be the set of *marked* paths where x is a bit-string of length $d+1$. We prepare $|\psi\rangle$ which is a superposition of non-backtracking walks from *entrance* to column $d+1$ as described before. Initially the overlap of $|\psi\rangle$ with $|G\rangle$ is $p = |\langle G|\psi\rangle|^2 = \frac{2^{d/3}}{2^d} = 2^{-2d/3}$. This is because the number of non-backtracking paths sampled from *exit* are $2^{d/3}$. Using Grover's search, $O(\frac{1}{\sqrt{p}})$ applications of oracle $\mathcal{O}_f$ are sufficient to find a marked path with high probability [2]. Thus the query complexity of this algorithm is $\tilde{\mathcal{O}}(2^{d/3})$. By the same argument in Lemma 1, the path obatined in the end is a shortest path from *entrance* to *exit*. □

Now, we are ready to prove the main theorem.

Theorem 1 *There is a polynomial speed-up, from $\tilde{\Theta}(2^{d/2})$ queries classicaly to $\tilde{\mathcal{O}}(2^{d/3})$ quantumly, for* Finding Path To Exit *problem in a welded tree of depth d.*

Proof In Theorem 2, we prove that the lower bound for any classical algorithm solving Finding Path To Exit is $\mathcal{O}(2^{d/2})$ queries. As mentioned before and described in Appendix 5.1.2, the $\mathcal{O}(\tilde{2^{d/2}})$ query algorithm by Ben-David [12] is hence classically optimal upto exponential factors. The theorem follows by Lemma 3 and above description. □

3.3 Small-space version of first quantum algorithm

In this section we describe a space efficient version of solving this collision problem. We focus on the usual setting of almost-random endofunctions H on 2^{d+2} binary words, where there is a significant proportion of collisions $H(w) = H(w')$ that encode an expected path between the *entrance* and *exit*. We then apply a classical cryptanalysis algorithm [13] to find these collisions using low memory.

Lemma 4 *The collision problem of finding the shortest path in welded trees of depth d between the initially known* entrance *and* exit *can be solved in $\mathcal{O}(2^{d/2})$ queries using $\mathcal{O}(d)$ bits.*

Proof We define the function E from binary words of length $d+2$ to the set of 2^{d+1} welding edges between the columns d and $d+1$. We fix an arbitrary order of the C colors for proper coloring.

For a binary word $w = w_0 w_1 \dots w_{d+1}$, we construct a non-backtracking path of $d+1$ edges as follows:

- If $w_0 = 0$, the path $\pi = \pi_0 \dots \pi_{d+2}$ starts from the *entrance*; otherwise, it starts from the *exit*.
- For $k = 1, \dots, d+1$, π_k is the vertex reached from π_{k-1} by the non-backtracking incident edge, minimal if $w_k = 0$ and maximal if $w_k = 1$.

Evaluating $E(w)$ requires $\mathcal{O}(d)$ queries to the oracle. We return the unoriented edge by sorting the labels:

$$E(w) := (\min(\mathsf{label}(\pi_d), \mathsf{label}(\pi_{d+1})), \max(\mathsf{label}(\pi_d), \mathsf{label}(\pi_{d+1}))) =: (e^{\min}(w), e^{\max}(w)).$$

By construction, this function produces 2^{d+1} collisions corresponding to pairs of paths of length $d+1$ from the *entrance* and *exit* that end at the same unoriented edge. From these words w and w' such that $E(w) = E(w')$, we can reconstruct an expected path from the *entrance* to the *exit*.

We then define an endofunction $H(w) := h(E(w))$ on binary words of length $d+2$, where the hash function h is given by:

$$h((e^{\min}, e^{\max})) := v^{\min} v^{\max},$$

where $v^{\min}$ is the prefix of length $\lceil (d+2)/2 \rceil$ of the ID $e^{\min}$ and $v^{\max}$ is the prefix of length $\lfloor (d+2)/2 \rfloor$ of the ID $e^{\max}$.

The hash function h introduces only a small number of "bad" collisions, where:

$$h((\mathsf{label}(v_1), \mathsf{label}(v_2))) = h((\mathsf{label}(v_3), \mathsf{label}(v_4)))$$

and $(v_1, v_2) \neq (v_3, v_4)$. These collisions arise due to the almost uniform sampling of label, which avoids the all-zero case 0^{2d}. Therefore, the collision probability is close to 2^{-d}, and if there is a collision $v_i = v_j$, the probability is approximately $2^{-d/2}$.

The expected number of "bad" collisions is bounded by:

$$E[\text{bad collisions}] \leq 2^{2d+2} \times 2^{-d} + 2^{d+1} \times 2^{-d/2} \leq 2^{d+2}.$$

Thus, the number of bad collisions is at most twice the number of good collisions, since there are fewer than 2^{2d+2} pairs of $((v_1, v_2), (v_3, v_4))$.

Given the almost independent and uniform sampling of the labels, the function H is nearly uniform, conditioned on the good collisions. Classical algorithms for collision detection [13] can then be applied. These algorithms require $\mathcal{O}(d)$ bits of memory and $\mathcal{O}(\sqrt{2^{d+2}})$ evaluations of H.

Each evaluation of H requires $\mathcal{O}(d \cdot 2^{d/2})$ queries to the oracle, but only $\mathcal{O}(d)$ bits are needed to store the last two vertices of the path π. Thus, $\mathcal{O}(d \cdot 2^{d/2})$ queries and $\mathcal{O}(d)$ space are sufficient to find a collision and, therefore, a path from the *entrance* to the *exit*. □

4 Classical algorithms with lesser memory

The previous section focused on the optimisation of the number of queries to the oracle. This section explores the trade-off when we add memory resources to the picture of optimisation. First, we consider a classical memory algorithm with the minimal number of bits $\mathcal{O}(d)$: the random walk that stores only the *id* of the current vertex. We then consider non-backtracking walks, a slight variant of the former, which store the previously visited vertex. This requires a constant number of additional bits, specifically $\lceil \ln_2(C) \rceil$, since a proper coloring with $C = 9$ exists for welded trees [3]. For both types of walks, we provide explicit expressions for the average number of queries.

The algorithm for a classical random walk simply stores the current *id* on $2d = \mathcal{O}(d)$ bits and then update it by the oracle, choosing one of the neighbors uniformly. This is the minimal space for an algorithm exploring the graph by the oracle without sampling random possible *id*s. The average number of queries starting from *entrance* and ending at *exit* corresponds to the hitting time which is well studied in the theory of random walks. A generic formula by Tetali [14] can be adapted to welded trees, giving the following lemma.

Lemma 5 *The average number of queries* $H_d(entrance, exit)$ *for a random walk* RW_d *starting from* entrance *of welded trees of depth* d *and ending at first visit of* exit *is*

$$H_d(entrance, exit) = \frac{4 \times 2^d - 3}{2^{d+1}} \times 2(3 \times 2^d - 2).$$

Proof More generally, the hitting time $H_d(u, v)$ is the expected length of a walk started at vertex u and ending at first visit of the vertex v in welded trees of depth d. The commuting time $C_d(u, v) = H_d(u, v) + H_d(v, u)$ is the sum of two hitting times. Tetali [14] shows that for any connected graph with m edges, including welded trees,

$$C_d(u, v) = 2mR(u, v)$$

where $R(u,v)$ is the "electric resistance" where the graph is interpreted as an electronic circuit where each edge has a resistance 1 then Kirhoff's rules defines $R(u,v)$.

In welded trees, the number of vertices is $2 \times (2^{d+1}-1)$ and any vertex has three incident edges, with one less for the two roots, so the number of edges is

$$m_d = \frac{3}{2}(2 \times (2^{d+1}-1)) - 1 = 2(3 \times 2^d - 2).$$

Moreover by symmetry of welded trees

$$H_d(entrance, exit) = H_d(exit, entrance) = C_d(entrance, exit)/2$$

so it only remains to evaluate the resistance $R_d(entrance, exit)$.

It appears that the welding permutation encoded by $\sigma^{[d,d+1]}$ does not change the resistance $R_d(entrance, exit)$. Indeed, a random walk in welded trees may be traced only at the level of columns as a random walk on $2d+2$ columns: for a vertex in column C_x with $x \leq d$, respectively $x \geq d+1$, has one neighbor in column C_{x-1}, respectively C_{x+1}, and two neighbors in column C_{x+1}, respectively C_{x-1}. Since the *entrance* and *exit* are alone in their respective columns C_0 and C_{2d+2}, the hitting time from C_0 to C_{d+2} for the random walk on columns corresponds to the hitting time $H_d(entrance, exit)$. Hence the choice of the welding permutation between columns d and $d+1$ does not modify the hitting time $H_d(entrance, exit)$.

To evaluate the resistance we choose the easily welded trees (see Appendix 5.1.1) i.e., $\sigma^{[d,d+1]}$ is the identity. This corresponds to highly self-similar series parallel circuit whose resistance can be recursively computed.

For $d = 0$, the welded trees have two vertices and two edges (in parallel) and by Kirchoff's law, the initial resistance is

$$R_0(entrance, exit) = \frac{1}{\frac{1}{1} + \frac{1}{1}} = \frac{1}{2}.$$

For $d > 0$, the welded trees of depth d may be seen as composed of two parallel circuits between *entrance* and exit. Each of those circuits compose in series, an edge from the *entrance* of resistance 1, the resistance R_{d-1} for a smaller welded tree of depth $d-1$, an again an edge to the *exit* of resistance 1. Hence by Kirchoff's law we have

$$R_d(entrance, exit) = \frac{1}{2 \times \frac{1}{1+R_{d-1}(entrance,exit)+1}} = 1 + \frac{R_{d-1}(entrance, exit)}{2}.$$

Solving this recurrence, we obtain

$$R_d(entrance, exit) = \frac{4 \times 2^d - 3}{2^{d+1}}$$

and applying Tetali's formula we obtain the expected exact formula for hitting time. □

In a random walk, remembering the color of the last edge used allows the walk to avoid stepping back to the previously visited vertex, by excluding it from the set of available neighbors at each step. This leads to the notion of non-backtracking walk that still uses $\mathcal{O}(d)$ bits with an additional $\lceil \ln_2(C) \rceil$ bits (where $C = 9$) . In this particular case, we obtain an exact formula by a more explicit computation.

Lemma 6 *The average number of queries $H_d^{nb}(entrance, exit)$ for a non-backtracking random walk RW_d^{nb} starting from* entrance *of welded trees of depth d and ending at first visit of* exit *is*

$$H_d^{nb}(entrance, exit) = \frac{2 \times 2^d - 1}{2^{d+1}} \times 2(3 \times 2^d - 2).$$

Proof We break any non-backtracking walk from *entrance* to *exit* into three parts: (1) the initial $(d+1)$ steps to reach a vertex in column $(d+1)$; (2) an arbitrary number of *fail paths*; and (3) the final *success path* to the exit. Each part corresponds to a set of paths W, analyzed using the probabilistic generating function:

$$L_W(x) := \sum_{w \in W} \mathsf{P}(w) x^{|w|},$$

where $\mathsf{P}(w)$ is the probability of path w, and $|w|$ its length. See [16] for background.

The first part includes 2^{d+1} paths from *entrance* to column $d+1$, each of probability $2^{-(d+1)}$, so the generating function is:

$$L_0(x) = 2^{d+1} \cdot 2^{-(d+1)} \cdot x^{d+1} = x^{d+1}.$$

The success path is a single d-step walk from column $d+1$ to *exit*, with probability 2^{-d}, yielding:

$$L_{\text{success}}(x) = \left(\frac{x}{2}\right)^d.$$

The middle part comprises repeated failed attempts: a walk from column $d+1$ to column $d+1+i < 2d+2$, followed by a descent to column d, further descent to column $d-j$ (possibly reaching $d-j=0$), and then an ascent back to $d+1$. The generating functions for the ascending and descending parts are:

$$L_{iv}(x) = \sum_{t=0}^{d-1} \left(\frac{1}{2}\right)^{t+1} x^{2t+1}, \quad L_v(x) = \left(\frac{1}{2}\right)^d x^{2d+1} + \sum_{t=0}^{d} \left(\frac{1}{2}\right)^{t+1} x^{2t+1},$$

where the first term in $L_v(x)$ accounts for the case where the walker returns to the *entrance* and must take the only other available edge. Thus, the generating function for a failed attempt is:

$$L_{\text{fail}}(x) = L_{iv}(x) \cdot L_v(x).$$

The complete generating function is:

$$L(x) = L_0(x) \cdot \left(\sum_{k \geq 0} \left(L_{\text{fail}}(x)\right)^k \right) \cdot L_{\text{success}}(x) = \frac{L_0(x) \cdot L_{\text{success}}(x)}{1 - L_{\text{fail}}(x)}.$$

After simplification:

$$L(x) = \frac{\left(x^2-2\right)^2 \left(\frac{x}{2}\right)^d}{\left(x^2 \left(\frac{x}{2}\right)^d - \left(\frac{x}{2}\right)^d - 1\right)\left(x^2 - x\left(\frac{x}{2}\right)^d + x - 2\right)}.$$

To compute the expected hitting time from *entrance* to *exit*, we differentiate $L(x)$ and evaluate at $x=1$:

$$\mathbb{E}[H_{\text{entrance,exit}}] = \left.\frac{dL(x)}{dx}\right|_{x=1} = (2^{d+1}-1) \cdot \left(\frac{3 \cdot 2^d - 2}{2^d}\right). \tag{4}$$

In the welded tree W, there are $m = 2(3 \cdot 2^d - 2)$ edges, so we can express the expected hitting time as:

$$\mathbb{E}[H_{\text{entrance,exit}}] = \left(\frac{2 \cdot 2^d - 1}{2^{d+1}}\right) \cdot m.$$

In this specific case, the factor $\left(\frac{2 \cdot 2^d - 1}{2^{d+1}}\right)$ can be viewed as a modified form of the effective resistance between *entrance* and *exit* for a non-backtracking walk on W. □

5 Classical complexity of Finding Path To Exit

5.1 Classical algorithm for upper bound

5.1.1 Exit finding in glued/easily-welded trees

In this section, we present in detail, a classical algorithm for exit finding in the glued trees or easily welded-trees (where $\sigma^{[d,d+1]}$ is the identity permutation in our notations) presented in [3] with a query complexity of $\Theta(d^2)$.

In other words up to permutations, a welded-trees graph is *easily welded* if the welding permutation is an involution, i.e., only made of cycles of size two, forming a double edge between pairs of leaves in distinct trees. In an easily welded-tree graph, one can easily detect that a vertex v is in (central) columns d or $d+1$ if and only if two distinct colored edges from v lead to the same neighbor. A non-backtracking walk from the root can track the length of the walk upto the central columns, and thus can check efficiently if a vertex is in column d. More precisely, a check of all the C colors allows to find all neighbors and the possible multiple occurences in them.

Algorithm 2 Classical algorithm finding *exit* with $\Theta(d^2)$ queries in glued trees.

Input: Easily welded tree (glued tree) W of depth d, oracle $\mathbf{\Omega}$ and the *id* of the *entrance*

Output: *id* of the *exit*.

```
procedure SearchExitInEasilyWeldedTrees(entrance, Ω)
    w ← A random non-backtracking walk of d steps starting from entrance
    prev_d ← last(w)
    while exit is not reached do
        v ← A random non-backtracking step from last(w) among unvisited edges
        w ← w ∪ {v}
        if last(w) is a vertex in column d then
            2k ← Number of vertices in w after prev_d                ▷ 2k ≥ 2
            w ← (w)_{1,...,ℓ(w)−k}                ▷ Remove the last k vertices of w.
            prev_d ← last(w)
        end if
    end while
    return last(w)
end procedure
```

The key in the analyis of this algorithm is to observe that the length $2k$ is strictly increasing during this algorithm since the bad choice of k vertices that made the walk to visit column d is replaced by the single remaining unvisited edge and thus is necessarly a good choice leading to column $d+k$ that will never be changed later. When $k = d+1$, it correponds to the column $2d+1$ containing only the *exit* vertex, so this algorithm ends on a walk w between *entrance* and *exit*. In the worst case, k is incremented only by 1 every time, so it visits about $d+1+\sum_{k=1}^{d+1} k = \Theta(d^2)$ vertices with the same number of calls to the classical oracle $\mathbf{\Omega}$.

5.1.2 Ben-David's optimal classical algorithm for Finding Exit

An algorithm attributed to Ben-David [12] has been identified in an online forum with a query complexity that matches the exponential order of the established lower bound. We present it in detail in this section for self containedness. It should also be noted that a similar algorithm along with an optimal lower bound was proven in the masters thesis of Rui Peng Liu [18].

Algorithm 3 Classical algorithm finding *exit* using $\tilde{\mathcal{O}}(2^{d/2})$ queries and $\tilde{\mathcal{O}}(2^{d/2})$ bits

Input: Welded tree W of depth d, oracle $\mathbf{\Omega}$ and the *id* of the *entrance*
Output: A shortest path from *entrance* to *exit* of the welded tree W.

```
procedure IsExit(id of vertex v)
    if v has only 2 neighbours and id(v) ≠ entrance then
        return Yes        ▷ Check for number of neighbours using Ω(id(v), c) for all colors c.
    end if
    return No
end procedure
procedure SearchExitClassically(entrance, Ω)
    C_⌈d/2⌉ ← Set of all vertices at distance ⌈d/2⌉ from entrance
    w ← A random non-backtracking walk of d steps starting from entrance
            ▷ We also store all the edges considered during the run of the algorithm
    prev_d ← last(w)
    while IsExit(last(w)) ≠ Yes do
        v ← A random non-backtracking step from last(w) among unvisited edges
        C'_⌊d/2⌋ ← Set of all vertices at distance ⌊d/2⌋ from last(w)
        if C_⌈d/2⌉ ∩ C'_⌊d/2⌋ ≠ ∅ then
            2k ← Number of vertices in w after prev_d                    ▷ 2k ≥ 2
            w ← (w)_{1,...,ℓ(w)−k}        ▷ Remove the last k vertices of w.
            prev_d ← last(w)
        end if
    end while
    return last(w)
end procedure
```

The correctness of this algorithm is inherited from the already discussed correctness of SearchExitInEasilyWeldedTrees (see also [3]). The single missing element is deciding if a vertex v belongs to column d. This element is obtained by the following equivalence: $\mathsf{last}(w)$ is in column d if and only if the intersection of $\mathcal{O}(2^{d/2})$ vertices of $C_{\lceil d/2\rceil}$ at distance $\lceil d/2\rceil$ from entrance and $\mathcal{O}(2^{d/2})$ vertices of $C'_{\lfloor d/2\rfloor}$ at distance $\lfloor d/2\rfloor$ of $\mathsf{last}(w)$ is a single vertex v' also along the single shortest path between entrance and $\mathsf{last}(w)$. In addition, after the first $d+1$ non-backtracking steps, the algorithm test only vertices at distance at least d from the entrance so the

equivalence is simplified by requiring just that the intersection of $C_{\lceil d/2 \rceil}$ and $C'_{\lfloor d/2 \rfloor}$ is non-empty since the cases of intersections of more then one vertex are avoided.

Regarding the query complexity, the initial computation of $C_{\lceil d/2 \rceil}$ requires $\Theta(2^{d/2})$ queries to the oracle. Each test that a vertex $\mathsf{last}(w)$ belongs to column d requires $\mathcal{O}(2^{d/2})$ queries. There are $\Theta(d^2)$ tests along the algorithm as in the easy version. Hence we obtain an algorithm with $\mathcal{O}(d^2 2^{d/2}) = \tilde{O}(2^{d/2})$ queries where tilde hides polynomial factors. Notice that sorting the vertices by ids in $C_{\lceil d/2 \rceil}$ and $C'_{\lfloor d/2 \rfloor}$ may add an additional factor d to the runtime complexity but none for the number of queries to the oracle.

In terms of memory, we just need to store the vertices of $C_{\lceil d/2 \rceil}$ and the last $C'_{\lfloor d/2 \rfloor}$ so the algorithm requires $\tilde{\mathcal{O}}(2^{d/2})$ bits of memory.

5.1.3 Lower memory version of Ben-David's algorithm

The optimal classical algorithm due to Ben-David [12], described in Algorithm 3, begins by exploring all $2^{\lceil d/2 \rceil}$ vertices in column $C_{\lceil d/2 \rceil}$.

To explore the time-space tradeoff, we consider a family of variants of this algorithm, parameterized by two values $0 \leq \gamma \leq \alpha \leq 1$: the modified algorithm explores only $2^{\gamma d}$ vertices in column $C_{\alpha d}$. The case $\gamma = \alpha = \frac{\lceil d/2 \rceil}{d} \approx \frac{1}{2}$ corresponds to the original Ben-David algorithm.

The space used is $\mathcal{O}(2^{\gamma d})$, mainly for storing the $2^{\gamma d}$ vertices. Thus, for small values of γ, the space required is small—but how does this impact the number of queries?

When $\gamma < \alpha$, the test to determine whether a vertex lies in column d may fail with probability $2^{(\gamma-\alpha)d}$. Therefore, we need to estimate the probability that the algorithm fails.

We assume that the $2^{\gamma d}$ vertices in column $C_{\alpha d}$ have already been sampled. Suppose we know a path from the *entrance* to column $d+1$, with an additional k edges leading toward the *exit* in the right subtree.

To find the $(k+1)^{\text{th}}$ correct edge, we must choose between two non-backtracking options leading to vertices v_1 and v_2. To determine whether a choice is wrong, we attempt to certify that one of the v_i leads in the incorrect direction.

To do this, we follow the same approach: for each vertex v_i, explore all 2^k non-backtracking walks of length k, which in the wrong case reach column C_d, and in the correct case reach some column C_y with $y > d+1$. For each such walk, we explore a ball of radius $(1-\alpha)d$, reaching a stored vertex in column $C_{\alpha d}$ with probability $2^{(\gamma-\alpha)d}$ in the wrong case, and otherwise none.

This certification attempt succeeds after $\mathcal{O}(2^k 2^{(1-\alpha)d})$ queries if we find a vertex in column d, with expected count $2^k 2^{(\gamma-\alpha)d}$, and thus success probability at most $\min(1, 2^{k+(\gamma-\alpha)d})$.

The probability of successfully finding a complete path to the *exit* for $k = 1, 2, \ldots, d$ is therefore bounded by:

$$\prod_{k=1}^{d} \min(1, 2^{k+(\gamma-\alpha)d}) = \prod_{k=1}^{(\alpha-\gamma)d} 2^{k+(\gamma-\alpha)d} = 2^{\binom{(\alpha-\gamma)d}{2} - ((\alpha-\gamma)d)^2} = 2^{-\binom{(\alpha-\gamma)d+1}{2}} \leq 2^{-(\alpha-\gamma)d^2}.$$

Hence, the algorithm must make $\mathcal{O}(2^{(\alpha-\gamma)d^2})$ attempts to succeed.

We now analyze the number of queries. For certificates in the case $k < (\alpha-\gamma)d$, we require $\mathcal{O}(2^k 2^{(1-\alpha)d})$ queries. For $k \geq (\alpha-\gamma)d$, the expected number of successful certifications becomes a positive constant after $\mathcal{O}(2^{(\alpha-\gamma)d})$ queries.

Summing over all d steps, we obtain:

$$\sum_{k=1}^{(\alpha-\gamma)d-1} 2^k 2^{(1-\alpha)d} + \sum_{k=(\alpha-\gamma)d}^{d} 2^{(\alpha-\gamma)d} 2^{(1-\alpha)d} \leq 2^{(1-\alpha)d} \left(2^{(\alpha-\gamma)d} + (1-\alpha-\gamma)d \cdot 2^{(\alpha-\gamma)d}\right),$$

which gives a total query complexity of $\tilde{\mathcal{O}}(2^{(1-\alpha)d+(\alpha-\gamma)d})$.

We thus obtain the following result:

Proposition 4 *The Ben-David algorithm when parameterized by $0 \leq \gamma \leq \alpha \leq 1$, performs $\mathcal{O}(2^{(1-\alpha)d+(\alpha-\gamma)d(d+1)})$ queries and uses $\tilde{\mathcal{O}}(2^{\gamma d})$ space.*

5.2 Classical lower bound for Finding Path To Exit

Any classical algorithm $\mathcal{A}$ solving the Finding Path To Exit problem relies on an oracle $f_c(u) = v$ that given the *id* of a vertex u and a color c returns the *id* of the vertex v adjacent to u along an edge colored c. We interpret such a query as an orientation from u to v of the edge (u, v) which is initially unoriented in the welded tree. It is safe to assume that the algorithm never performs the non-informative reverse query $f_c(v) = u$ since any algorithm that performs such a query can be modified with more memory but without extra queries to one that does not perform any non-informative queries. Hence the record of queries may be interpreted as a partial orientation of edges of the welded trees.

A *collision* in this partial orientation is either a pair of distinct edges $((u, v), (w, v))$ both oriented toward the same vertex v or an oriented edge (u, v) where v is either *entrance* or *exit*. Any run of a classical randomised algorithm $\mathcal{A}$ starts with a *collision-free phase* that ends with the first occurence of a collision. Collisions in the general case corresponds to the detection of cycle used in the previous lower bounds of [3] and [9]. The polynomial classical algorithm for Finding Exit problem in glued trees in [3], where the welding in made up of duplicated pairs of edges (u, v), relies on the detection of collisions along these edges.

During the collision-free phase, the algorithm is essentially performing a traversal of the edges of a binary tree from its root but after the first collision, the behaviour can be different and more difficult to analyse. To simplify the analysis even after the first collision, we consider a weaker oracle $f_c^{free}(\cdot)$ that prevents the detection of collision during the run of the algorithm. The possibility of detection is postponed until after the execution of the algorithm. This oracle takes as input a full non-backtracking path $p = (c_i)_{i=0,\dots k}$ defined by the sequence of edge colors from the *entrance* to a vertex u. The output $f_c^{free}(p) = \mathsf{label}(v, (p, c))$ where $\mathsf{label}(v, (p, c))$ is an new *id* everytime we query the same vertex v with different paths p. Hence v appears potentially in as many copies as the (infinite) number of paths from *entrance* to v so that the algorithm

can not detect any collision of two edges just by the fact they have endpoints of same *id*. There exists a map Φ (not accessible to the algorithm) that allows to recover the vertex v from any related label $\Phi(\mathsf{label}(v,p)) = v$, hence detecting the collisions in the run a posteriori. This weaker oracle for any algorithm is similar to an assumption on oracle used by Hastings in [19] to avoid detection of cycles.

Lemma 7 *For any randomized algorithm $\mathcal{A}$ that makes queries to an oracle Ω, the distribution of the number of queries made during the collision-free phase is the same whether $\Omega = f_c(\cdot)$ or $\Omega = f_c^{free}(\cdot)$.*

Proof While there is no collision, in both cases the algorithm collects *id*s of vertices that are all distinct. Hence we can not distinguish the behaviour of the two oracles. Since in addition both oracles are based on the same randomly sampled welded tree (see section 2), the distributions of the number of queries are equal. □

Hence we lower bound the expected number of queries in the collision-free phase for oracle $f_c(\cdot)$ by computing it for the case of $f_c^{free}(\cdot)$. Then our analysis relies on the following bounds on the expected number of edges between column C_x and $C_{x\pm1}$ for all possibles pairs.

Lemma 8 *Any algorithm $\mathcal{A}$ with the collision-free oracle $f_c^{free}(\cdot)$ performing at most $2^{\alpha d}$ queries, where $\alpha \in [0, 1/2)$, generates an expected number of at most*

(*a*) $2^{\alpha d}$ *edges from column x to $x+1$ for each $x \in \{0, 1, \dots d\}$.*
(*b*) $2^{\alpha d+d-x}$ *edges from column x to column $x+1$ for each $x \in \{d+1, \dots 2d\}$,*
(*c*) $2^{\alpha d+x-d-1}$ *edges from column x to column $x-1$ for each $x \in \{1, \dots d\}$.*
(*d*) $(2d+2-x)2^{\alpha d+d-x}$ *edges from column x to column $x-1$ for each $x \in \{d+1, \dots 2d+1\}$.*

Proof (*a*) Each discovered edge requires at least one query to the oracle.

(*b*) For $x \geq d+1$, any discovered edge from column x to column $x+1$ comes from an initial vertex in column $d+1$ and then exactly $x-d$ queries toward the *exit* vertex (not shared with other such discovered edges). The number of initial vertices (in column $d+1$) discovered by an edge between columns d and $d+1$ is bounded by $2^{\alpha d}$. Hence there are at most $2^{\alpha d}$ non-backtracking paths of length $x-d$ starting from such vertex. Each of these $2^{\alpha d}$ has a probability 2^{d-x} to ends in column C_{x+1} by a final edge from column C_x to column C_{x+1} since among the two non-backtracking steps exactly one is toward the exit. Hence the expected number of edges in this case is bounded by $2^{\alpha d+d-x}$.

(*c*) This case is symmetric to the case (*b*) where the initial vertices are now in column d, discovered by a query from column $d+1$. The length of the paths ending by the expected edge from column C_x to C_{x-1} is now $d+1-x$. Hence the expected number of edges is bounded by $2^{\alpha d+x-d-1}$.

(*d*) Any edge from column C_x to column C_{x-1} requires a path from column C_{d+1} to column C_{x+k} for some k with $0 \leq k \leq 2d+1-x$ then a path of length $k+1$ toward the entrance. For each k, our upper bound comes from each path from column $d+1$ to column

$x+k$ that may define up to 2^k paths leading to a query from column x to column $x-1$. Using bounds of case (b) the expected number $|S|$ of edges from column x to $x-1$ is bounded as follows

$$E(|S|) \leq \sum_{k=0}^{2d+1-x} 2^k \cdot 2^{\alpha d+d-x-k} \leq (2d+1-x+1)2^{\alpha d+d-x}.$$

□

With those bounds on the expected number of edges between various columns, we can upper bound the probability of collision for any algorithm using the collision-free oracle.

Lemma 9 *For any $\alpha \in [0,1/2)$, the expected number of collisions after $2^{\alpha d}$ queries in any algorithm using collision-free oracle $f_c^{free}(\cdot)$ is bounded by $3(2+6d)d2^{(2\alpha-1)d}$.*

Proof A collision occurs in a column C_x from one edge e_1 coming from column C_y where $y \in \{x \pm 1\}$ and in general a second edge e_2 coming from column C_z where $z \in \{x \pm 1\}$. We use a union bound on those $2+6d$ triplets (x,y,z). Our maximal bound in Lemma 8 is $2^{\alpha d}$ in the case (a) and all other cases, $(b),(c),(d)$, appears to be bounded by $3d2^{\alpha d-|d-x|}$ where we introduce the absolute value $|d-x|$. Moreover the targeted column C_x is all cases contains at least $2^{d-|d-x|}$ vertices. Our proof uses union bound on those $2+6d$ cases where expected number of collisions is uniformly bounded by $3d2^{(2\alpha-1)d}$.

First we consider a case where at least one of the edges e_1 and e_2 does not belong to case (a). Hence the expected number of pairs, with one edge from C_y to C_x and one edge from C_z to C_x, is bounded by $2^{\alpha d} \times 3d2^{\alpha d-|d-x|}$. For each pair, the probability that it leads to a collision is bounded by $2^{|d-x|-d}$. Hence the expected number of collision in this case is bounded by $2^{\alpha d} \times 3d2^{\alpha d-|d-x|} \times 2^{|d-x|-d} = 3d2^{(2\alpha-1)d}$.

Second, we consider triplets where both edges belongs to case (a). For $x < d$, it appears that there is no collision by design since we are dealing with the edges from the tree rooted at entrance. Hence it remains only the case $x = d$ that corresponds to the detection of collisions in column C_{d+1} among the welding edges. The number of pairs of edges is bounded by $2^{2\alpha d}$. Each pair has probability 2^{-d} to be a collision in this column C_{d+1} with 2^d vertices. Hence the expected number of collision is $2^{(2\alpha-1)d} \leq 3d2^{(2\alpha-1)d}$.

Finally, we consider the collisions at entrance, under case (c) that specializes to $2^{\alpha d-d}$ and exit, under case (b) that specialize also to $2^{\alpha d-d}$. Since $2^{\alpha d-d} \leq 3d2^{(2\alpha-1)d}$ we have the expected uniform bound in each of the $2+6d$ cases. □

This probability must remain bounded below by a constant independent of d in order to find the *exit* with constant probability. This, in turn, implies our expected lower bound on the number of queries during the collision-free phase. Since this collision-free phase is a must for any algorithm solving Finding Exit problem, we obtain our expected lower bound.

Theorem 2 *Any classical randomized algorithm that makes $\mathcal{O}(2^{\alpha d})$ queries to the oracle, where $\alpha \in [0,1/2)$, cannot solve the Finding Path To Exit problem with probability greater than $\tilde{\mathcal{O}}(2^{(2\alpha-1)d})$.*

Proof By contradiction, let $\mathcal{A}$ be an algorithm that solves FINDING EXIT problem with constant probability with an expected number of queries $2^{\alpha d}$ where $\alpha < 1/2$. It means that this algorithm has a constant probability $P > 0$ to find a collision within $2^{\alpha d}$ queries. According to the combination of Lemma 7 and Lemma 9, it means that for any d, $P \leq 3(2+6d)d2^{(2\alpha-1)d}$ which is a contradiction. □

6 Conclusion

The main result of this work is a quantum algorithm in $\tilde{\mathcal{O}}(2^{d/3})$ queries to the oracle to find a path from the *entrance* to the *exit* in welded trees and a collection of other classical algorithms with a perspective of time and space trade-offs. We also formalise the folklore lower bound $\tilde{\Omega}(2^{d/2})$ following the works of [3, 9, 10, 12] thereby proving a the first polynomial speedup of our quantum algorithm over classical randomised algorithms to the best of our knowledge. Even though this is not a resolution of Aaronson's original problem, we belive it is a step towards it. As a first step, it would be interesting to ask if it possible to solve FINDING PATH TO EXIT in $\mathcal{O}(2^{d/c})$ queries for some $c > 3$? A natural follow up question would be to ask for a lower bound for any quantum algorithm that involves finding a collision among two partial paths as in our case. It would also be interesting to find the exact expression of lower bound for the class of *rooted* quantum algorithms as described in the paper by Childs et al. [11]. We suspect that it should be $\tilde{\mathcal{O}}(2^{d/24})$. Moreover a lower bound for finding a path from *entrance* to *exit* with constant positive probability for a quantum algorithm would help resolve the exact query complexity of a quantum algoritm.

References

[1] Shor, P.W.: Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM Journal on Computing **26**(5), 1484–1509 (1997) https://doi.org/10.1137/S0097539795293172 https://doi.org/10.1137/S0097539795293172

[2] Grover, L.K.: A fast quantum mechanical algorithm for database search (1996). https://arxiv.org/abs/quant-ph/9605043

[3] Childs, A.M., Cleve, R., Deotto, E., Farhi, E., Gutmann, S., Spielman, D.A.: Exponential algorithmic speedup by a quantum walk. In: Proceedings of the Thirty-fifth Annual ACM Symposium on Theory of Computing, pp. 59–68 (2003)

[4] Jeffery, S., Zur, S.: Multidimensional quantum walks. In: Proceedings of the 55th Annual ACM Symposium on Theory of Computing, pp. 1125–1130 (2023)

[5] Belovs, A.: Global Phase Helps in Quantum Search: Yet Another Look at the Welded Tree Problem (2024) arXiv:2404.19476 [quant-ph]

[6] Li, G., Li, L., Luo, J.: Recovering the original simplicity: succinct and deterministic quantum algorithm for the welded tree problem. In: Proceedings of the 2024

Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 2454–2480 (2024). SIAM

[7] Gilyén, A., Hastings, M.B., Vazirani, U.: (sub)exponential advantage of adiabatic quantum computation with no sign problem. In: Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing. STOC 2021, pp. 1357–1369. Association for Computing Machinery, New York, NY, USA (2021). https://doi.org/10.1145/3406325.3451060 . https://doi.org/10.1145/3406325.3451060

[8] Ben-David, S., Childs, A.M., Gilyén, A., Kretschmer, W., Podder, S., Wang, D.: Symmetries, graph properties, and quantum speedups. SIAM Journal on Computing **53**(6), 20–36820415 (2024) https://doi.org/10.1137/23M1573975

[9] Fenner, S.A., Zhang, Y.: A note on the classical lower bound for a quantum walk algorithm. arXiv preprint quant-ph/0312230 (2003)

[10] Aaronson, S.: Open problems related to quantum query complexity. ACM Transactions on Quantum Computing **2**(4), 1–9 (2021)

[11] Childs, A.M., Coudron, M., Gilani, A.S.: Quantum algorithms and the power of forgetting. arXiv preprint arXiv:2211.12447 (2022)

[12] (pseudo), S.: Reply to The randomized query complexity of the conjoinded trees problem, https://cstheory.stackexchange.com/questions/25279/the-randomized-query-complexity-of-the-conjoined-trees-problem (2014)

[13] Oorschot Paul C., J., W.M.: Parallel collision search with cryptanalytic applications. Journal of cryptology (1999)

[14] Prasad, T.: Random walks and the effective resistance of networks. Journal of Theoretical Probability (1990)

[15] Brassard, G., Høyer, P., Mosca, M., Tapp, A.: Quantum amplitude amplification and estimation. American Mathematical Society (2002). https://doi.org/10.1090/conm/305/05215 . http://dx.doi.org/10.1090/conm/305/05215

[16] Flajolet, P., Sedgewick, R.: Analytic Combinatorics. Cambridge University Press, Cambridge, ??? (2009). https://doi.org/10.1017/CBO9780511801655 . https://doi.org/10.1017/CBO9780511801655

[17] Brassard, G., Høyer, P., Tapp, A.: Quantum cryptanalysis of hash and claw-free functions (invited paper). In: LATIN'98: Theoretical Informatics (Campinas, 1998). Lecture Notes in Comput. Sci., vol. 1380, pp. 163–169. Springer, ??? (1998). https://doi.org/10.1007/BFb0054319 . https://doi.org/10.1007/BFb0054319

[18] Liu, R.P.: Some Upper and Lower Bounds regarding Query Complexity. University of Waterloo (2017). http://hdl.handle.net/10012/12245

[19] Hastings, M.B.: The Power of Adiabatic Quantum Computation with No Sign Problem. Quantum **5**, 597 (2021) https://doi.org/10.22331/q-2021-12-06-597