Multidimensional Quantum Walks

Stacey Jeffery
CWI & QuSoft
Amsterdam, The Netherlands
smjeffery@gmail.com

Sebastian Zur
CWI & QuSoft
Amsterdam, The Netherlands
sebastian.zur@cwi.nl

ABSTRACT

While the quantum query complexity of k-distinctness is known to be $O(n^{\frac{3}{4}-\frac{1}{4}\frac{1}{2^k-1}})$ for any constant $k \geq 4$ [Belovs, FOCS 2012], the best previous upper bound on the time complexity was $\widetilde{O}(n^{1-1/k})$. We give a new upper bound of $\widetilde{O}(n^{\frac{3}{4}-\frac{1}{4}\frac{1}{2^k-1}})$ on the time complexity, matching the query complexity up to polylogarithmic factors. In order to achieve this upper bound, we give a new technique for designing quantum walk search algorithms, which is an extension of the electric network framework. We also show how to solve the welded trees problem in $O(n)$ queries and $O(n^2)$ time using this new technique, showing that the new quantum walk framework can achieve exponential speedups.

CCS CONCEPTS

• **Mathematics of computing** → **Graph algorithms**; • **Theory of computation** → **Quantum query complexity**.

KEYWORDS

quantum algorithms, quantum random walk, phase estimation, element distinctness, welded trees, superpolynomial speedup

ACM Reference Format:
Stacey Jeffery and Sebastian Zur. 2023. Multidimensional Quantum Walks. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing (STOC '23), June 20–23, 2023, Orlando, FL, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3564246.3585158

1 INTRODUCTION

In the problem of *element distinctness*, the input is a list of n integers, and the output is a bit indicating whether the integers are all distinct, or there exists a pair of integers that are the same, called a *collision*. This problem has been studied as a fundamental problem in query complexity, but also for its relationship to other more practical problems, such as sorting, or *collision finding*, which is similar, but one generally assumes there are many collisions and one wants to find one. In the worst case, element distinctness requires $\Theta(n)$ classical queries [2].

The first quantum algorithm to improve on this was a $O(n^{3/4})$ query algorithm [14], which is a variation of an optimal quantum algorithm for collision finding [13], whose main technique is amplitude amplification [12]. The algorithm of [14] could also be implemented time efficiently, in $\widetilde{O}(n^{3/4})$ steps, with a log factor overhead from storing large subsets of the input in a sorted data structure. This was later improved to $O(n^{2/3})$ queries, and $\widetilde{O}(n^{2/3})$ time by Ambainis [3], which is optimal [1]. Ambainis' algorithm has been modified to solve other problems in various domains, from k-sum [18], to path finding in isogeny graphs [20, 26]. Moreover, this algorithm was a critical step in our understanding of quantum query complexity, and quantum algorithms in general, as the algorithm used a new technique that was later generalized by Szegedy into a generic speedup for random walk search algorithms of a particular form [25].

For any constant integer $k \geq 2$, the problem k-*distinctness* is to decide if an input list of integers contains k copies of the same integer. When $k = 2$, this is exactly element distinctness. Ambainis [3] actually gave a quantum algorithm for k-distinctness for any $k \geq 2$, with query complexity $O(n^{1-1/(k+1)})$, and time complexity $\widetilde{O}(n^{1-1/(k+1)})$. For $k \geq 3$, Belovs gave an improved quantum query upper bound of $O(n^{3/4-\frac{1}{4}\frac{1}{2^k-1}})$ [8], however, this upper bound was not constructive. Belovs proved this upper bound by exhibiting a dual adversary solution, which can be turned into a quantum algorithm that relies on controlled calls to a particular unitary. This unitary can be implemented in one query, but actually implementing this algorithm requires giving an efficient circuit for the unitary, which is not possible in general. This is analogous to being given a classical table of values, but no efficient circuit description. While it seems reasonable to guess that the time complexity of k-distinctness should not be significantly higher than the query complexity – what could one possibly do aside from querying and sorting well chosen sets of inputs? – the problem of finding a matching time upper bound was open for ten years.

In the meantime, lower bounds of $\Omega(n^{\frac{3}{4}-\frac{1}{2k}})$ for $k \geq 3$ [15] and $\Omega(n^{\frac{3}{4}-\frac{1}{4k}})$ for $k \geq 4$ [24] were exhibited. Progress was also made for the $k = 3$ case. Two simultaneous works, [10] and [19] (published together as [11]), gave a $\widetilde{O}(n^{5/7})$ time upper bound for 3-distinctness. Ref. [10] achieved this bound using a generalization of Szegedy's quantum walk framework, called the *electric network framework*. Ref. [19] used the MNRS quantum walk framework [23], and could also be generalized to give a slight improvement on the time upper bound to $\widetilde{O}(n^{1-1/k})$ for any $k > 3$ [21].

In this work, we give an upper bound of $\widetilde{O}(n^{\frac{3}{4}-\frac{1}{4}\frac{1}{2^k-1}})$ on the time complexity of k-distinctness, matching the best known query upper bound up to polylogarithmic factors. We do this using ideas from Belovs' query upper bound in a new framework for quantum walk algorithms, the *multidimensional quantum walk framework*, which is an extension of the electric network framework – the most general of the quantum walk frameworks [6].We give a high-level overview of this extension in Section 2.

Quantum walk search frameworks are important because they allow one to design a quantum algorithm by first designing a classical random walk algorithm of a particular form, which can be compiled into an often faster quantum algorithm. While quantum walk frameworks make it extremely easy to design quantum algorithms, even without an in-depth knowledge of quantum computing, as evidenced by their wide application across domains, the major drawback is that they can achieve at most a quadratic speedup over the best classical algorithm. This is because a quantum walk search algorithm essentially takes a classical random walk algorithm, and produces a quantum algorithm that is up to quadratically better.

This drawback does not hold for the multidimensional quantum walk framework. We give a quantum algorithm in our framework that solves the *welded trees* problem in $O(n)$ queries and $O(n^2)$ time, which is an exponential speedup over the classical lower bound of $2^{\Omega(n)}$ [17]. While a $\mathrm{poly}(n)$ quantum algorithm based on continuous-time quantum walks was already known, this proof-of-concept application shows that our framework is capable of exponential speedups. We emphasize that unlike the quantum walk search frameworks mentioned here that give generic speedups over classical random walk algorithms, continuous-time quantum walks are not easily designed and analysed, and their applications have been limited (with some exceptions based on converting quantum walk search algorithms into continuous-time quantum walks, such as [5]). Our multidimensional quantum walk framework, as a generalization of the electric network framework, is in principal similarly easy to apply, but with the potential for significantly more dramatic speedups.

2 QUANTUM WALKS

We give a brief overview of previous work on quantum walk search algorithms, with sufficient detail to understand, at a high level, the improvements we make, before describing these improvements at the end of this section.

The first (discrete) quantum walk search framework is due to Szegedy [25], and is a generalization of the technique used by Ambainis in his element distinctness algorithm [3]. The framework can be described in analogy to a classical random walk algorithm that first samples an initial vertex according to the stationary distribution π of some random walk (equivalently, reversible Markov process) P, and repeatedly takes a step of the random walk by sampling a neighbour of the current vertex, checking each time if the current vertex belongs to some *marked set* M. Let $HT(P, M)$ be the hitting time, or the expected number of steps needed by a walker starting from π to reach a vertex in M. If S is the cost of sampling from π, U is the cost of sampling a neighbour of any vertex, C is the cost of checking if a vertex is marked, and H is an upper bound on $HT(P, M)$ assuming $M \neq \emptyset$, then this classical algorithm finds a marked vertex with bounded error in complexity:

$$O(\mathsf{S} + H(\mathsf{U} + \mathsf{C})).$$

Szegedy showed that given such a P and M, if S is the cost of coherently[1] sampling from π, i.e. generating $\sum_u \sqrt{\pi(u)}|u\rangle$, and U is the cost of generating, for any u, the superposition over its neighbours $\sum_v \sqrt{P_{u,v}}|v\rangle$, then there is a quantum algorithm that detects if $M \neq \emptyset$ with bounded error in complexity:

$$O(\mathsf{S} + \sqrt{H}(\mathsf{U} + \mathsf{C})).$$

This result was extended to the case of *finding* a marked vertex, rather than just *detecting* a marked vertex in [4]. This framework, and subsequent related frameworks have been widely applied, because this is a very simple way to design a quantum algorithm.

Belovs generalized this framework to the *electric network framework*, by allowing the initial state to be $|\sigma\rangle = \sum_u \sqrt{\sigma(u)}|u\rangle$ for *any* distribution σ, analogous to starting a random walk in some arbitrary initial distribution. Then if S_σ is the cost to generate $|\sigma\rangle$, there is a quantum algorithm that detects a marked vertex with bounded error in complexity:

$$O(\mathsf{S}_\sigma + \sqrt{C}(\mathsf{U} + \mathsf{C})),$$

where C is a quantity that may be the same, or much larger than the hitting time of the classical random walk starting at σ. For example, if $\sigma = \pi$, then $C = H$ as above, but when σ is supported on a single vertex s, and $M = \{t\}$, C is the *commute time* from s to t [16], which is the expected number of steps needed to get from s to t, and then back to s. If the hitting time from s to t is the same as the hitting time from t to s, this is just twice that hitting time. However, in some cases the hitting time from t to s may be significantly larger than the hitting time from s to t.

A second incomparable quantum walk search framework that is similarly easy to apply is the MNRS framework [23]. Loosely speaking, this is the quantum analogue of a classical random walk that does not check if the current vertex is marked at every step, but rather, only after sufficiently many steps have been taken so that the current vertex is independent of the previously checked vertex. Ref. [6] extended the electric network framework to be able to *find* a marked vertex, and also showed that the MNRS framework can be seen as a special case of the resulting framework. Thus, the finding version of the electric network framework captures all quantum walk search frameworks in one unified framework.

We now discuss, at a high level, how a quantum walk search algorithm works – particularly in the electric network framework (but others are similar)[2]. We will suppose for simplicity that σ is supported on a single vertex s, and either $M = \emptyset$ or $M = \{t\}$. Fix a graph G, possibly with weighted edges, such that $s, t \in V(G)$. It is simplest if we imagine that G is bipartite, so let $V(G) = V_{\mathcal{A}} \cup V_{\mathcal{B}}$ be a bipartition, with $s \in V_{\mathcal{A}}$. Let G' be the graph G with a single extra vertex v_0 that is not part the bipartition. This new vertex v_0 is connected to s, and connected to t if and only if $t \in M$. For $u \in V_{\mathcal{A}}$, define *star states*:

$$|\psi_\star^{G'}(u)\rangle = \sum_{v \in V_{\mathcal{B}} \cup \{v_0\} : \{u,v\} \in E(G')} \sqrt{\mathsf{w}_{u,v}}|u, v\rangle,$$

where $\mathsf{w}_{u,v}$ is the weight of the edge $\{u, v\}$. If we normalize this state, we get $\sum_v \sqrt{P_{u,v}}|u, v\rangle$, where P is the transition matrix of the random walk on G'. For $v \in V_{\mathcal{B}}$, define:

$$|\psi_\star^{G'}(v)\rangle = \sum_{u \in V_{\mathcal{A}} \cup \{v_0\} : \{u,v\} \in E(G')} \sqrt{\mathsf{w}_{u,v}}|u, v\rangle.$$

[1]Technically the classical S and U might be different from the quantum ones, but in practice they are often similar.

[2]We discuss the classic construction of such algorithms, without modifications that were more recently made in [4] and [6] to not only detect, but find.

Let

$$\mathcal{A} := \operatorname{span}\{|\psi_\star^{G'}(u)\rangle : u \in V_{\mathcal{A}}\}\ ,\ \mathcal{B} := \operatorname{span}\{|\psi_\star^{G'}(v)\rangle : v \in V_{\mathcal{B}}\}.$$

Then a quantum walk algorithm works by performing phase estimation of the unitary

$$U_{\mathcal{A}\mathcal{B}} := (2\Pi_{\mathcal{A}} - I)(2\Pi_{\mathcal{B}} - I)$$

on initial state $|s, v_0\rangle$ to some sufficiently high precision – this precision determines the complexity of the algorithm. Let us consider why this algorithm can distinguish $M = \emptyset$ from $M = \{t\}$.

First suppose $M = \{t\}$. Assume there is a path from s to t in G (otherwise a random walk from s will never find t), which means there is a cycle in G' containing the edge (v_0, s), obtained by adding (t, v_0) and (v_0, s) to the st-path in G. We can define a cycle state for a cycle $u_1, \dots, u_d = u_1$ as:

$$\sum_{i=1}^{d-1} \frac{|e_{u_i,u_{i+1}}\rangle}{\sqrt{\mathsf{w}_{u_i,u_{i+1}}}} \text{ where } |e_{u,v}\rangle := \begin{cases} |u,v\rangle & \text{if } (u,v) \in V_{\mathcal{A}} \times V_{\mathcal{B}} \\ -|v,u\rangle & \text{if } (u,v) \in V_{\mathcal{B}} \times V_{\mathcal{A}}. \end{cases}$$

A cycle state is orthogonal to all star states: if the cycle goes through a vertex u, it is supported on 2 of the edges adjacent to u: one contributing -1 because it goes into u, and the other $+1$ because it comes out of u. Thus a cycle state is in the $(+1)$-eigenspace of $U_{\mathcal{A}\mathcal{B}}$. If there is a cycle that uses the edge (v_0, s), then it has non-zero overlap with the initial state $|s, v_0\rangle$, and so the initial state has non-zero overlap with the $(+1)$-eigenspace of $U_{\mathcal{A}\mathcal{B}}$, and so the phase estimation algorithm will have a non-zero probability of outputting a phase estimate of 0. The shorter the cycle (i.e. the shorter the st-path) the greater this overlap is relative to the size of the cycle state. We can make a similar argument if we take not just a single st-path in G, but a superposition of paths called an st-flow. Then the *energy* of this flow controls the probability of getting a phase estimate of 0. The minimum energy of a unit flow from s to t is called the *effective resistance* between s and t, denoted $\mathcal{R}_{s,t}(G)$.

On the other hand, suppose $M = \emptyset$. Then we claim that

$$|s, v_0\rangle = \sum_{u \in V_{\mathcal{A}}} |\psi_\star^{G'}(u)\rangle - \sum_{v \in V_{\mathcal{B}}} |\psi_\star^{G'}(v)\rangle \in \mathcal{A} + \mathcal{B} = (\mathcal{A}^\perp \cap \mathcal{B}^\perp)^\perp.$$

This means that our initial state has no overlap with the $(+1)$-eigenspace of $U_{\mathcal{A}\mathcal{B}}$, which is exactly $(\mathcal{A} \cap \mathcal{B}) \oplus (\mathcal{A}^\perp \cap \mathcal{B}^\perp)$, so if we could do phase estimation with infinite precision, the probability we would measure a phase estimate of 0 would be 0. Our precision is not infinite, but using a linear algebraic tool called the *effective spectral gap lemma*, we can show that precision proportional to

$$\left\| \sum\nolimits_{u \in V_{\mathcal{A}}} |\psi_\star^{G'}(u)\rangle \right\|^2 = \sum\nolimits_{e \in G'} \mathsf{w}_e =: \mathcal{W}(G)$$

is sufficient.

Combining these two analyses for the $M = \{t\}$ and $M = \emptyset$ case yield (in a non-obvious way) that approximately $\sqrt{\mathcal{R}\mathcal{W}}$ steps of the quantum walk is sufficient, if $\mathcal{R}$ is an upper bound on $\mathcal{R}_{s,t}(G)$ whenever $M = \{t\}$, and $\mathcal{W}$ is an upper bound on $\mathcal{W}(G)$ whenever $M = \emptyset$. A nice way to interpret this is that the quantity $\mathcal{R}_{s,t}(G)\mathcal{W}(G)$ is equal to the *commute time* from s to t – the expected number of steps a random walker starting from s needs to reach t, and then return to s. For a discussion of how to interpret this quantity in the case of more general σ and M, see [6].

The Multidimensional Quantum Walk Framework: We extend this algorithm in two ways:

Edge Composition To implement the unitary $U_{\mathcal{A}\mathcal{B}}$, we perform a mapping that acts, for any $u \in V_{\mathcal{A}}$, as $|u, 0\rangle \mapsto |\psi_\star^{G'}(u)\rangle$ (up to normalization), and a similar mapping for $v \in V_{\mathcal{B}}$. Loosely speaking, what this means is that we have a labelling of the edges coming out of u, and some way of computing (u, v) from (u, i), where v is the i-th neighbour of u. If this computation costs $\mathsf{T}_{u,i}$ steps, then it takes $O(\max_{u,i} \mathsf{T}_{u,i})$ steps to implement $U_{\mathcal{A}\mathcal{B}}$. However, in case this cost varies significantly over different u, i, we can do much better. We show how we can obtain a unitary with polylogarithmic cost, and essentially consider, in the analysis of the resulting algorithm, a quantum walk on a modified graph in which an edge $\{u, v\}$, where v is the i-th neighbour of u, is replaced by a path of length $\mathsf{T}_{u,i}$. A similar thing was already known for *learning graphs*, when a transition could be implemented with $\mathsf{T}_{u,i}$ *queries* [9]. This is an extremely useful, if not particularly surprising, feature of the framework, which we use in our application to k-distinctness.

Alternative Neighbourhoods The more interesting way we augment the electric network framework is to allow the use of *alternative neighbourhoods*. In order to generate the star state of a vertex u, a superposition of the edges coming out of u, one must, in some sense, know the neighbours of u, as well as their relative weights. In certain settings, the algorithm will know that the star state for u is one of a small set of easily preparable states $\Psi_\star(u) = \{|\psi_\star^1(u)\rangle, |\psi_\star^2(u)\rangle, \dots\}$, but computing precisely which one of these is the correct state would be computationally expensive. In that case, we include all of $\Psi_\star(u)$ when constructing the spaces $\mathcal{A}$ and $\mathcal{B}$. In the case when $M = \emptyset$, the analysis is the same – by increasing $\mathcal{A} + \mathcal{B}$, we have only made the analysis easier. However, in the case $M \neq \emptyset$, the analysis has become more constrained. For the analysis of this case, we used a circulation, because it is orthogonal to all star states. However, now there are some extra states in $\mathcal{A} + \mathcal{B}$, and we need to take extra care to find a circulation that is also orthogonal to these.

The new alternative neighbourhoods technique is best understood through examples, of which we shortly describe two. We first remark on the unifying idea from which both these techniques follow.

If we let $\{|\psi_\star(u)\rangle\}_{u \in V}$ be *any* set of states, we can make a graph G on V by letting u and v be adjacent if and only if $\langle \psi_\star(u) | \psi_\star(v) \rangle \neq 0$. Then, if this graph is bipartite, and we can reflect around the span of each state individually, we can reflect around $\operatorname{span}\{|\psi_\star(u)\rangle : u \in V\}$. Quantum walk search algorithms can be seen as a special case of this, where we additionally exploit the structure of the graph to analyse the complexity of this procedure. One way of viewing alternative neighbourhoods is that we extend this reasoning to the case where we have *spaces* $\{\operatorname{span}\{\Psi_\star(u)\}\}_{u \in V}$, each of which we can efficiently reflect around, and G is now a bipartite graph encoding the overlap of the *spaces*, hence the qualifier *multidimensional.*

Edge composition also exploits this picture. We can define a sequence of subspaces $\{\Psi_t^{u,v}\}_{t=1}^{\mathsf{T}_{u,i}}$ that only overlap for adjacent t, and such that the subroutine computing $|v, j\rangle$ from $|u, i\rangle$ can be seen as moving through these spaces. Now the overlap graph of all these spaces will look like G, except with each edge (u,v) replaced by a path of length $\mathsf{T}_{u,i}$. For a more detailed explanation and for examples of such overlap graphs, see [22]. Before moving on to our examples, we comment that unlike the finding version of the electric network framework [6], our extension does not allow one to find a marked vertex, but only to detect if there is one or not. We leave extending our framework to finding as future work.

3 WELDED TREES

We motivate the alternative neighbourhoods modification by an application to the welded trees problem [17]. In the welded trees problem, the input is an oracle O_G for a graph G with $s,t \in V(G) \subset \{0,1\}^{2n}$. Each of s and t is the root of a full binary tree with 2^n leaves, and we connect these leaves with a pair of random matchings. This results in a graph in which all vertices except s and t have degree 3, and s and t each have degree 2. Given a string $u \in \{0,1\}^{2n}$, the oracle O_G returns $\perp$ if $u \notin V(G)$, which is true for all but at most a 2^{-n+2} fraction of strings, and otherwise it returns a list of the 2 or 3 neighbours of u. We assume $s = 0^{2n}$, so we can use s as our starting point, and the goal is to find t, which we can recognize since it is the only other vertex with only 2 neighbours. The classical query complexity of this problem is $2^{\Omega(n)}$ [17]. Intuitively that is because this problem is set up so that a classical algorithm has no option but to do a random walk, starting from s, until it hits t. However, this takes $2^{\Omega(n)}$ steps, because wherever a walker is in the graph, the probability of moving towards the centre, where the leaves of the two trees are connected, is twice the probability of moving away from the centre, towards s or t. So a walker quickly moves from s to the centre, but then it takes exponential time to escape to t.

While we know there is a quantum algorithm that solves this problem in poly(n) queries[3] to O_G [17], if we try to reproduce this result in the electric network framework, we will get an exponential-time algorithm, essentially because the total weight of the graph is exponential.

Suppose we could add weights to the edges of G, so that at any vertex u, the probability of moving towards the centre or away from the centre were the same: that is, if w is the weight on the edge from u to its *parent*, then the other two edges should have weight $\mathsf{w}/2$. This would already be very helpful for a classical random walk, however, a bit of thought shows that this is not possible to implement. By querying u, we learn the labels of its three neighbours, v_1, v_2, v_3, which are random $2n$-bit strings, but we get no indication which is the parent. However, we know that the correct star state in the weighted graph that we would like to be able to walk on is proportional to one of the following:

$$|u,v_1\rangle + \frac{1}{2}|u,v_2\rangle + \frac{1}{2}|u,v_3\rangle,$$
$$|u,v_2\rangle + \frac{1}{2}|u,v_1\rangle + \frac{1}{2}|u,v_3\rangle,$$
$$|u,v_3\rangle + \frac{1}{2}|u,v_1\rangle + \frac{1}{2}|u,v_2\rangle.$$

Thus, we add all three states (up to some minor modifications) to $\Psi_\star(u)$, which yields an algorithm that can learn any bit of information about t in $O(n)$ queries. By composing this with the Bernstein-Vazirani algorithm we can find t. For an detailed exhibition of this algorithm, see[22].

We emphasize that our application to the welded trees problem does not use the edge composition technique. It would be trivial to embed any known exponential speedup in our framework by simply embedding the exponentially faster quantum algorithm in one of the edges of the graph, but we are able to solve the welded trees problem using only the alternative neighbourhoods idea.

[3]The best previous query complexity was $O(n^{1.5})$ [7], although it is likely that continuous time quantum walks could also be used to solve this problem in $O(n)$ queries.

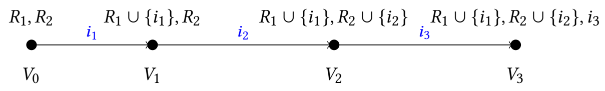
R_1, R_2
i_1
$R_1 \cup \{i_1\}, R_2$
i_2
$R_1 \cup \{i_1\}, R_2 \cup \{i_2\}$
i_3
$R_1 \cup \{i_1\}, R_2 \cup \{i_2\}, i_3$
V_0
V_1
V_2
V_3

Figure 1: A sample path from V_0 to V_3 in our first attempt at a quantum walk for 3-distinctness. The indices shown in blue can be seen to label the edges.

4 3-DISTINCTNESS

We first describe an attempt at a quantum walk algorithm for the simpler case of 3-distinctness, how it fails, and how the Multidimensional Quantum Walk Framework comes to the rescue. While our result for $k = 3$ is not new, our generalization to $k > 3$ is, and the case of $k = 3$ is already sufficient to illustrate our techniques. For the details of how our framework tackles k-distinctness, see [22]. Formally, the problem of 3-distinctness is: given a string $x \in [q]^n$, output a 1 if and only if there exist distinct $a_1, a_2, a_3 \in [n]$ such that $x_{a_1} = x_{a_2} = x_{a_3}$. We make the standard simplifying assumptions (without loss of generality) that if such a 3-collision exists, it is unique, and moreover, there is an equipartition $[n] = A_1 \cup A_2 \cup A_3$ such that $a_1 \in A_1$, $a_2 \in A_2$ and $a_3 \in A_3$.

We now describe a graph that will be the basis for a quantum walk attempt. A vertex v_{R_1,R_2} is described by a pair of sets $R_1 \subset A_1$ and $R_2 \subset A_2$. v_{R_1,R_2} stores these sets, as well as input-dependent *data* consisting of the following:

- Queried values for all of R_1: $D_1(R) := \{(i, x_i) : i \in R_1\}$.
- Queried values for those elements of R_2 that have a match in R_1:
$$D_2(R) := \{(i_1, i_2, x_{i_1}) : i_1 \in R_1, i_2 \in R_2, x_{i_1} = x_{i_2}\}.$$

By only keeping track of the values in R_2 that have a match in R_1, we save the cost of initially querying the full set R_2. The vertices will be in 4 different classes, for some parameters r_1 and r_2 with $r_1 \ll r_2$:

$$V_0 = \{v_{R_1,R_2} : |R_1| = r_1, |R_2| = r_2\}$$
$$V_1 = \{v_{R_1,R_2} : |R_1| = r_1 + 1, |R_2| = r_2\}$$
$$V_2 = \{v_{R_1,R_2} : |R_1| = r_1 + 1, |R_2| = r_2 + 1\}$$
$$V_3 = \{v_{R_1,R_2,i_3} : |R_1| = r_1 + 1, |V_2| = r_2 + 1, i_3 \in A_3\}.$$

The vertices $v_{R_1,R_2,i_3} \in V_3$ are just like the vertices in V_2, except there is an additional index $i_3 \in A_3$ stored. We connect vertices in V_ℓ and $V_{\ell+1}$ in the obvious way: $v_{R_1,R_2} \in V_\ell$ is adjacent to $v_{R_1',R_2'} \in V_{\ell+1}$ if and only if $R_1 \subseteq R_1'$ and $R_2 \subseteq R_2'$ (exactly one of these inclusions is proper); and $v_{R_1,R_2} \in V_2$ is adjacent to $v_{R_1,R_2,i_3} \in V_3$ for any $i_3 \in A_3$ (see Figure 1).

We say a vertex $v_{R_1,R_2,i_3} \in V_3$ is marked if $a_1 \in R_1$, $a_2 \in R_2$, and $a_3 = i_3$, where (a_1, a_2, a_3) is the unique 3-collision. Thus a quantum walk that decides if there is a marked vertex or not decides 3-distinctness.

We imagine a quantum walk that starts in a uniform superposition over V_0. To construct this initial state, we first take a uniform superposition over all sets R_1 of r_1 indices, and query them. Next we take a uniform superposition over all sets R_2 of size r_2, but rather than query everything in R_2, we search for all indices in R_2 that have a match in R_1. This saves us the cost of querying all r_2 elements of R_2, which is important because we will set r_2 to be larger than the total complexity we aim for (in this case, $r_2 \gg n^{5/7}$), so we could not afford to spend so much time. However, we do not only care about query complexity, but also the total time spent on non-query operations, so we also do not want to spend time writing down the set R_2, even if we do not query it, which is the first problem with this approach:

Problem 1: Writing down R_2 would take too long.

The fix for Problem 1 is rather simple: we will not let R_2 be a uniform random set of size r_2. Instead, we will assume that A_2 is partitioned into m_2 blocks, each of size $n/(3m_2)$, and R_2 will be made up of $t_2 := 3m_2r_2/n$ of these blocks. This also means that when we move from V_1 to V_2, we will add an entire block, rather than just a single index. The main implication of this is that when we move from V_1 to V_2, we will have to search the new block of indices that we are adding to R_2 for any index that collides with R_1. This means that transitions from V_1 to V_2 have a non-trivial cost, n^ε for some small constant ε, unlike all other transitions, which have polylogarithmic cost. Naively we would incur a multiplicative factor of n^ε on the whole algorithm, but we avoid this because the edge composition technique essentially allows us to only incur the cost n^ε on the edges that actually incur this cost, and not on every edge in the graph. Otherwise, our solution to Problem 1 is technical, but not deep, and so we gloss over Problem 1 and its solution for the remainder of this high-level synopsis. This is the only place we use the edge composition part of the framework in our applications, but we suspect it can be used in much more interesting ways.

Moving on, in order to take a step from a vertex $v_{R_1,R_2} \in V_0$ to a vertex $v_{R_1\cup\{i_1\},R_2} \in V_1$, we need to select a uniform new index i_1 to add to R_1, and then also update the data we store with each vertex. That means we have to query i_1 and add (i_1, x_{i_1}) to $D_1(R)$, which is simple, and can be done in $O(\log n)$ basic operations as long as we use a reasonable data structure to store $D_1(R)$; and we also have to update $D_2(R)$ by finding anything in R_2 that collides with i_1. Since R_2 has not been queried, this latter update would require an expensive search, which we do not have time for, so we want to avoid this. However, if we do not search R_2 for any i_2 such that $x_{i_2} = x_{i_1}$, then whenever we add some i_1 that has a match in R_2, the data becomes incorrect, and we have introduced what is referred to in [8] as a *fault*. This is a serious issue, because if i_1 is the unique index in R_1 such that there exists $i_2 \in R_2$ with $x_{i_1} = x_{i_2}$ but this is not recorded in $D_2(R)$, then i_1 is "remembered" as having been added after i_2. That is, the resulting vertex does not only depend on $R_1 \cup \{i_1\}, R_2$, but on i_1 as well. For quantum interference to happen, it is crucial that when we are at a vertex v, the state does not remember anything about how we got there.

Problem 2: When we add i_1 to R_1 without searching for a match in R_2, we may introduce a *fault*.

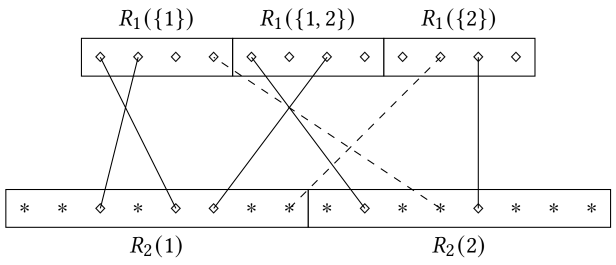

Figure 2: The data we keep track of for a vertex v_{R_1,R_2}. $\diamond$ represents a queried index. $*$ represents an index whose query value is not stored. We only store the query value of an index in $R_2(s)$ if it collides with something in $R_1(\{s\}) \cup R_1(\{1,2\})$, shown here by a solid line. If $i_2 \in R_2(1)$ collides with some value in $R_1(\{2\})$, shown here by a dashed line, we do not record that, and do not store x_{i_2}.

Our handling of this is inspired by the solution to an analogous problem in the query upper bound of [8]. We partition R_1 into three sets: $R_1(\{1\})$, $R_1(\{2\})$, and $R_1(\{1,2\})$; and R_2 into two sets $R_2(1)$ and $R_2(2)$. Then $D_2(R)$ will only store collisions (i_1, i_2, x_{i_1}) such that $x_{i_1} = x_{i_2}$ if $i_1 \in R_1(S)$ and $i_2 \in R_2(s)$ for some $s \in S$. This is shown in Figure 2.

Now when we add i_1 to R_1, we have three choices: we can add it to $R_1(\{1\})$, $R_1(\{2\})$, or $R_1(\{1,2\})$. Importantly, at least one of these choices does not introduce a fault. To see this, suppose there is some $i_2 \in R_2$ such that $x_{i_1} = x_{i_2}$. We claim there can be at most one such index, because otherwise there would be a 3-collision in $A_1 \cup A_2$, and we are assuming the unique 3-collision has one part in A_3. This leads to three possibilities:

Type 1: $i_2 \in R_2(2)$, in which case, adding i_1 to $R_1(\{1\})$ does not introduce a fault.

Type 2: $i_2 \in R_2(1)$, in which case, adding i_1 to $R_1(\{2\})$ does not introduce a fault.

Type 0: There is no such i_2, in which case, adding i_1 to $R_1(\{1\})$ or $R_1(\{2\})$ or $R_1(\{1,2\})$ does not introduce a fault.

We modify the graph so that we first move from $v_{R_1,R_2} \in V_0$ to $v_{R_1,R_2,i_1} \in V_0^+$ by selecting a new $i_1 \in A_1 \setminus R_1$, and then move from v_{R_1,R_2,i_1} to $v_{R_1\cup\{i_1\},R_2} \in V_1$ – here there are three possibilities for $R_1 \cup \{i_1\}$, depending on to which of the three parts of R_1 we add i_1. However, we will only add i_1 to a part of R_1 that does not introduce a fault. Thus, a vertex v_{R_1,R_2,i_1} in V_0^+ has one edge leading back to V_0, and either one or three edges leading forward to V_1, as shown in Figure 3.

On its own, this is not a solution, because for a given v_{R_1,R_2,i_2}, in order to determine its type, we would have to search for an $i_2 \in R_2$ such that $x_{i_1} = x_{i_2}$, which is precisely what we want to avoid. However, this is exactly the situation where the alternative neighbourhood technique is useful. For all $u \in V_0^+$, we will let $\Psi_\star(u)$ contain all three possibilities shown in Figure 3, of which exactly one is the correct state. We are then able to carefully construct a flow that is orthogonal to all three states, in our analysis. The idea is that all incoming flow from v must leave along the edge $(u, v^{\{1\}})$ so that the result is a valid flow in case of Type 1. However, in order to be a valid flow in case of Type 2, all incoming flow from v

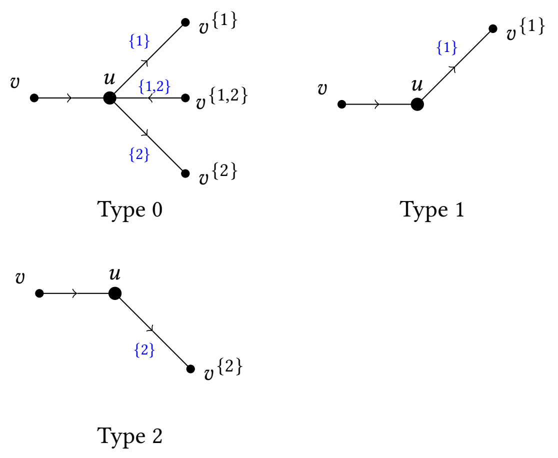

Figure 3: The possible neighbourhoods of $u = v_{R_1,R_2,i_1} \in V_0^+$, depending on the type of vertex. $v^S \in V_1$ is obtained from v by adding i_1 to $R_1(S)$. The backwards neighbour $v = v_{R_1,R_2} \in V_0$ is always the same.

must leave along the edge $(u, v^{\{2\}})$. But now to ensure that we also have a valid flow in case of Type 0, we must have negative flow on the edge $(u, v^{\{1,2\}})$, or equivalently, flow from $v^{\{1,2\}}$ to u. This is indicated by the arrows on the edges in Figure 3.

Model of Computation: Our k-distinctness algorithm works in the same model as previous k-distinctness algorithms, which we try to make more explicit than has been done in previous work. In addition to arbitrary 1- and 2-qubit gates, we assume *quantum random access* to a large *quantum* memory (QRAM). This version of QRAM is fully quantum, whereas some previous works have used "QRAM" to refer to classical memory that can be read in superposition by a quantum machine. For more details, see [22].

ACKNOWLEDGEMENTS

We thank Arjan Cornelissen and Maris Ozols for discussions on the early ideas of this work; and Simon Apers, Aleksandrs Belovs, Shantanav Chakraborty, Andrew Childs, Frédéric Magniez and Ronald de Wolf for helpful comments and discussions about these results. This work is supported by ERC STG grant 101040624-ASC-Q, NWO Klein project number OCENW.Klein.061, and ARO contract no W911NF2010327. SJ is a CIFAR Fellow in the Quantum Information Science Program.

REFERENCES

[1] Scott Aaronson and Yaoyun Shi. 2004. Quantum lower bounds for the collision and the element distinctness problems. *Journal of the ACM* 51, 4 (2004), 595–605. https://doi.org/10.1145/1008731.1008735
[2] Miklós Ajtai. 2005. A non-linear time lower bound for Boolean branching programs. *Theory of Computing* 1 (2005), 149–176.
[3] Andris Ambainis. 2007. Quantum Walk Algorithm for Element Distinctness. *SIAM Journal on Computing* 37, 1 (2007), 210–239. https://doi.org/10.1137/S0097539705447311 Earlier version in FOCS'04. arXiv: quant-ph/0311001
[4] Andris Ambainis, András Gilyén, Stacey Jeffery, and Martins Kokainis. 2020. Quadratic Speedup for Finding Marked Vertices by Quantum Walks. In *Proceedings of the 52nd ACM Symposium on the Theory of Computing (STOC).* 412–424. https://doi.org/10.1145/3357713.3384252 arXiv: 1903.07493
[5] Simon Apers, Shantanav Chakraborty, Leonardo Novo, and Jérémie Roland Roland. 2021. Quadratic speedup for spatial search by continuous-time quantum walk. (2021). https://doi.org/10.48550/arXiv.2112.12746 arXiv: 2112.12746
[6] Simon Apers, András Gilyén, and Stacey Jeffery. 2020. A Unified Framework of Quantum Walk Search. In *Proceedings of the 38th Symposium on Theoretical Aspects of Computer Science (STACS).* 6:1–6:13. https://doi.org/10.4230/LIPIcs.STACS.2021.6 arXiv: 1912.04233
[7] Yosi Atia and Shantanav Chakraborty. 2021. Improved Upper Bounds for the Hitting Times of Quantum Walks. *Physical Review A* 104 (2021), 032215. https://doi.org/10.1103/PhysRevA.104.032215 arXiv: 2005.04062
[8] Aleksandrs Belovs. 2012. Learning-graph-based quantum algorithm for k-distinctness. In *Proceedings of the 53rd IEEE Symposium on Foundations of Computer Science (FOCS).* 207–216. https://doi.org/10.1109/FOCS.2012.18 arXiv: 1205.1534
[9] Aleksandrs Belovs. 2012. Span programs for functions with constant-sized 1-certificates. In *Proceedings of the 44th ACM Symposium on the Theory of Computing (STOC).* 77–84. https://doi.org/10.1145/2213977.2213985
[10] Aleksandrs Belovs. 2013. Quantum walks and electric networks. (2013). arXiv: 1302.3143
[11] Aleksandrs Belovs, Andrew M. Childs, Stacey Jeffery, Robin Kothari, and Frédéric Magniez. 2013. Time-Efficient Quantum Walks for 3-Distinctness. In *Proceedings of the 40th International Colloquium on Automata, Languages, and Programming (ICALP).* 105–122. https://doi.org/10.1007/978-3-642-39206-1_10
[12] Gilles Brassard, Peter Høyer, Michele Mosca, and Alain Tapp. 2002. Quantum Amplitude Amplification and Estimation. In *Quantum Computation and Quantum Information: A Millennium Volume.* Contemporary Mathematics Series, Vol. 305. AMS, 53–74. https://doi.org/10.1090/conm/305/05215 arXiv: quant-ph/0005055
[13] Gilles Brassard, Peter Høyer, and Alain Tapp. 1997. Quantum Algorithm for the Collision Problem. *ACM SIGACT News* 28 (1997), 14–19. arXiv: quant-ph/9705002
[14] Harry Buhrman, Christoph Dürr, Mark Heiligman, Peter Høyer, Frédéric Magniez, Miklos Santha, and Ronald de Wolf. 2005. Quantum Algorithms for Element Distinctness. *SIAM Journal on Computing* 34, 6 (2005), 1324–1330. https://doi.org/10.1137/S0097539702402780 Earlier version in CCC'01. arXiv: quant-ph/0007016
[15] Mark Bun, Robin Kothari, and Justin Thaler. 2018. The polynomial method strikes back: Tight quantum query bounds via dual polynomials. In *Proceedings of the 50th ACM Symposium on the Theory of Computing (STOC).* https://doi.org/10.1145/3188745.3188784 arXiv: 1710.09079
[16] Ashok K. Chandra, Prabhakar Raghavan, Walter L. Ruzzo, Roman Smolensky, and Prasoon Tiwari. 1996. The electrical resistance of a graph captures its commute and cover times. *Computational Complexity* 6, 4 (1996), 312–340. https://doi.org/10.1007/BF01270385
[17] Andrew M. Childs, Richard Cleve, Enrico Deotto, Edward Farhi, Sam Gutmann, and Daniel A. Spielman. 2003. Exponential algorithmic speedup by a quantum walk. In *Proceedings of the 35th ACM Symposium on the Theory of Computing (STOC).* 59–68. https://doi.org/10.1145/780542.780552 arXiv: quant-ph/0209131
[18] Andrew M. Childs and Jason M. Eisenberg. 2003. Quantum algorithms for subset finding. (2003). arXiv: quant-ph/0311038
[19] Andrew M. Childs, Stacey Jeffery, Robin Kothari, and Frédéric Magniez. 2013. A Time-Efficient Quantum Walk for 3-Distinctness Using Nested Updates. (2013). arXiv: 1302.7316
[20] Craig Costello, Patrick Longa, and Michael Naehrig. 2016. Efficient algorithms for supersingular isogeny Diffie-Hellman. In *Advances in Cryptology (CRYPTO 2016).* 572–601.
[21] Stacey Jeffery. 2014. *Frameworks for Quantum Algorithms.* Ph. D. Dissertation. University of Waterloo. Available at http://uwspace.uwaterloo.ca/handle/10012/8710.
[22] Stacey Jeffery and Sebastian Zur. 2022. Multidimensional Quantum Walks, with Application to k-Distinctness. *arXiv preprint arXiv:2208.13492* (2022).
[23] Frédéric Magniez, Ashwin Nayak, Jérémie Roland, and Miklos Santha. 2011. Search via Quantum Walk. *SIAM Journal on Computing* 40, 1 (2011), 142–164. https://doi.org/10.1137/090745854 Earlier version in STOC'07. arXiv: quant-ph/0608026
[24] Nikhil S. Mande, Justin Thaler, and Shuchen Zhu. 2020. Improved Approximate Degree Bounds for k-Distinctness. In *Proceedings of the 15th Conference on the Theory of Quantum Computation, Communication, and Cryptography (TQC),* Vol. 158. 2:1–2:22. https://doi.org/10.4230/LIPIcs.TQC.2020.2 arXiv: 2002.08389
[25] Mario Szegedy. 2004. Quantum speed-up of Markov chain based algorithms. In *Proceedings of the 45th IEEE Symposium on Foundations of Computer Science (FOCS).* 32–41. https://doi.org/10.1109/FOCS.2004.53 arXiv: quant-ph/0401053
[26] Seiichiro Tani. 2009. Claw finding algorithms using quantum walk. *Theoretical Computer Science* 410, 50 (2009), 5285–5297.

Received 2022-11-07; accepted 2023-02-06