Low-gate Quantum Golden Collision Finding

Samuel Jaques[1] and André Schrottenloher[2]

[1] Department of Materials, University of Oxford, United Kingdom
samuel.jaques@materials.ox.ac.uk
[2] Inria, Paris, France
andre.schrottenloher@inria.fr

Abstract. The golden collision problem asks us to find a single, special collision among the outputs of a pseudorandom function. This generalizes meet-in-the-middle problems, and is thus applicable in many contexts, such as cryptanalysis of the NIST post-quantum candidate SIKE.
The main quantum algorithms for this problem are memory-intensive, and the costs of quantum memory may be very high. The quantum circuit model implies a linear cost for random access, which annihilates the exponential advantage of the previous quantum collision-finding algorithms over Grover's algorithm or classical van Oorschot-Wiener.
Assuming that quantum memory is costly to access but free to maintain, we provide new quantum algorithms for the golden collision problem with high memory requirements but low gate costs. Under the assumption of a two-dimensional connectivity layout, we provide better quantum parallelization methods for generic and golden collision finding. This lowers the quantum security of the golden collision and meet-in-the-middle problems, including SIKE.

Keywords: Quantum cryptanalysis, golden collision search, quantum walks, SIKE.

1 Introduction

Quantum computers have a significant advantage in attacking some widely-used public-key cryptosytems. In light of the continuing progress on quantum architectures, the National Institute of Standards and Technology (NIST) launched a standardization process for new primitives [29], which is still ongoing.

The new cryptosytems proposed rely on generic problems that are believed to be hard for quantum computers. That is, contrary to the discrete logarithm problem in abelian groups, or to the factorization of integers, they should not admit polynomial-time quantum algorithms. However, an exponential algorithm could be relevant if the non-asymptotic cost is low enough, so these attacks still require careful analysis.

In this paper, we study quantum algorithms for the *golden collision search* problem. In the context of the NIST call, these algorithms can be applied in a generic key-recovery of the NIST candidate SIKE (non-commutative supersingular isogeny-based key encapsulation) [20,2,15]. They can also be used in some lattice attacks [3].

Golden Collision Search. We have access to a function $h : X \to X$ that has collisions, *i.e.* pairs of inputs with the same output value. Collisions happen randomly, but (at most) one of them is *golden* and we wish to retrieve it.

Classically, the most time-efficient method is to retrieve a whole lookup table for h, sort by output value and look at all collisions. However, this incurs a massive cost in random-access memory. A study with limited memory was done in [2]. The authors concluded that the most efficient method was van Oorschot-Wiener's distinguished point technique [30]. In the context of SIKE, they noticed that the proposed parameters offered even more security when accounting for memory limits.

Quantum Circuits. In this work, we study quantum algorithms written in the *quantum circuit model*, which abstracts out the physical architecture. The computation is a sequence of basic *quantum gates* applied to a pool of *qubits*, *i.e.* two-level quantum systems. The time complexity in this model is thought of as the number of operations applied, that is, the number of quantum gates.

The best quantum algorithm for golden collision search is Ambainis' algorithm [4], with time $\widetilde{\mathcal{O}}(N^{2/3})$ if $|X| = N$, matching a query lower bound of $\mathcal{O}(N^{2/3})$ [1]. However, it suffers from a heavy use of *quantum random access* to massive amounts of quantum memory, and does not fare well under depth constraints.

In this paper, we dismiss quantum RAM and use only the *baseline* circuit model, as in [22]. We consider that a memory access to R qubit registers requires $\Theta(R)$ quantum gates. With this restriction, we design new quantum algorithms for golden collision search.

Metrics. We consider the two metrics of *gate count* (G) and *depth-width* product (DW) emphasized in [22]. The first one assumes that the *identity gate* costs 0, meaning we can leave as many qubits idle for as long as we want. This happens *e.g.* if the decoherence time of individual qubits, when no gates are applied, can be prolonged to arbitrary lengths at a fixed cost. The second one considers instead that the identity gate costs 1. This happens *e.g.* if error-correction must be performed at each time step, on all qubits. In addition, since we consider quantum circuits at a large scale, we account for locality constraints with a model of a two-dimensional grid with nearest-neighbour interactions only.

Contributions. We first optimize for the *gate count* in Section 3. We rewrite van Oorschot-Wiener collision search as a random walk so that we can obtain a quantum analogue in the MNRS *quantum walk* framework. If h is a single gate evaluated in time 1, our algorithm gives a G-cost of $\widetilde{\mathcal{O}}(N^{6/7})$. Next, we give another algorithm that searches for *distinguished points* with Grover's search. These two methods achieve the exact same complexity.

In Section 4, we give a parallel version of our prefix-based walk, and a parallel multi-Grover search algorithm that improves over [8]. This gives the G-cost and DW-cost of our algorithms under depth constraints, improving on the counts of [22].

NIST defined five security levels relative to the hardness of breaking symmetric cryptographic schemes, possibly with some depth limitation. Three of these levels compare to a Grover search, which is well-understood. Two of them compare to a *collision search* (this time, not golden). We extend our study of SIKE parameters to these two security levels. For this purpose, we analyze the collision search algorithm of [14], which gives the lowest gate count and depth-width product when memory accesses are of linear cost. In Section 5, we provide its best parallelization to date. Finally, in Section 6, we show that the SIKE parameters have lower quantum security than claimed in [22], but they still meet the NIST security levels claimed in [20].

2 Preliminaries

2.1 Computational model

For classical computers, we imagine a parallel random access machine with a shared memory. Costs are in RAM operations, with access to the memory having unit cost.

We write quantum algorithms in the *quantum circuit model* [28]. In order to give meaningful cost estimates of quantum circuits, we use the memory peripheral model of [22]. We model the quantum computer as a peripheral of a classical parallel random access machine, which acts on the quantum computer using the Clifford+T gate set. We define two cost metrics:

- The G-cost of an algorithm is the number of gates, each of which costs one RAM operation to the classical controller. Here, we assume that error correction is *passive*, meaning that once a qubit is in a particular state, we incur no cost to maintain that state indefinitely.
- The DW-cost is the depth-width product of the circuit. Here, error correction is *active*. At each time step, the classical controller must act on each qubit of the circuit, even if the qubit was idle at this point.

Connectivity. The standard quantum circuit model assumes no connectivity restriction on the qubits. Two-qubit gates can be applied on any pair of qubits without overhead. In Section 3 we do not refer to the connectivity, but in Section 4, this layout plays a role, so we consider the two following alternatives: a 2-dimensional grid with nearest-neighbor connectivity (*local*) or no restriction (*nonlocal*). It is shown in [8] that the nonlocal case can be emulated by any network, with a multiplicative overhead that is the time to sort the network.

Quantum Memory Models. Many quantum algorithms use the "qRAM" model, in which the access *in superposition* to the elements in memory is a cheap operation. But the cost of qRAM is unclear at the moment[3]. This model can be restricted

[3] See [17,6] for the "bucket-brigade" architecture, which still requires $\Theta(R)$ gates for a memory access to R bits of memory.

to *quantum-accessible classical memory* (QRACM, see [24, Section 2]), while the best time complexities for golden collision search [4,32] require QRAQM, that is, the memory accessed contains quantum states.

Both QRAQM and QRACM can be constructed in the quantum circuit model with Clifford+T gates. The caveat is that, for R bits of memory, both will require $\Theta(R)$ gates for each memory access. QRAQM will necessarily require R qubits, while QRACM could sequentially simulate the access with $\text{poly}(R)$ qubits, and R classical memory. In this work we use only the standard quantum circuit model, so each memory access incurs this large gate cost. In other words, we assume a world in which quantum circuits are scalable, but qRAM is not cheap.

2.2 Problem Description

We focus on the golden collision problem (Problem 2.1), although it is possible to go back and forth to *element distinctness* and *claw-finding*.

Problem 2.1 (Golden collision finding). Let $h : X \to X$ be a random function and $g : X \times X \to \{0, 1\}$ be a *check*. The function h has collisions: pairs $x, y \in X$ such that $h(x) = h(y)$. The function g takes a collision as input, and outputs 1 for a certain set of $\mathcal{O}(1)$ collisions, which we call the *golden collisions*. Given access to h and g, find a golden collision.

In many instances there will be a unique golden collision. If h is not pseudo-random, we pick a random function $f : X \to X$ and assume that $f \circ h$ is pseudo-random. In practice, this holds if h does not have a serious restriction on its outputs.

Problem 2.2 (Element distinctness). Given $h : \{0, 1\}^n \to \{0, 1\}^n$, determine if h is a permutation or not.

This reduces to golden collision search by composing with a random function; the check function is to apply just h and check for the true collision.

Problem 2.3 (Claw-finding). Given $f : \{0, 1\}^n \to \{0, 1\}^m$ and $g : \{0, 1\}^n \to \{0, 1\}^m$, where we assume $m \geq 2n$, find a *claw*: a pair x, y such that $f(x) = g(y)$.

If we construct a random function from $\{0, 1\}^m$ to $\{0, 1\} \times \{0, 1\}^n$, then we can act on $\{0, 1\} \times \{0, 1\}^n$ with f and g by sending $(0, x)$ to $f(x)$ and $(1, x)$ to $g(x)$. The claw becomes a golden collision for the concatenation of these two functions, where we check collisions by checking if they are caused because $f(x) = g(y)$ or by our random function.

Notations. We define $N = 2^n$, the size of the domain and range of h. We let H denote the cost of evaluating h and G the cost of g. In cases where we need to distinguish between the gates, depth, or width of evaluating h, we will use subscripts of G, D, and W, respectively. The memory size is denoted R. Memory is typically counted in n-bit registers that represent inputs or outputs of h.

2.3 Previous Works

We assume that h and g can be evaluated in $\mathsf{poly}(n)$ time. Classically, the query complexity is $\Theta(N)$, since one must at least query every element to find the golden collision. One algorithm to achieve this is to construct a table for all $x, h(x)$, sort the table by the value of $h(x)$, and check each collision. The most prominent practical algorithm for golden collision finding is due to van Oorschot and Wiener [30]. Their method is simple and parallelizes perfectly. With R elements of memory, it requires $\mathcal{O}(N^{3/2}/R^{1/2})$ operations, which is asymptotically optimal for $R = N$.

Buhrman *et al.* [13] give a quantum algorithm in time $\widetilde{\mathcal{O}}(N^{3/4})$ and $\mathcal{O}(N^{1/2})$ memory for claw-finding and element distinctness. This algorithm uses Grover search as a subroutine, and can be recovered by the optimization program of [27]. Ambainis [4] gives a quantum walk algorithm with $\widetilde{\mathcal{O}}(N^{2/3})$ quantum time, with a query complexity of $\mathcal{O}(N^{2/3})$, which is optimal [1]. Tani provided a claw-finding version [32].

However, Buhrman *et al.*'s, Ambainis' and Tani's algorithms require respectively $\mathcal{O}(N^{1/2})$ and $\mathcal{O}(N^{2/3})$ qubits with cheap quantum random access. If random access to a memory of size R requires $\Theta(R)$ gates, then the *gate complexity* of these algorithms is actually $\widetilde{\mathcal{O}}(N^{4/3})$, although they can be reparameterized to reach $\widetilde{\mathcal{O}}(N)$. Grover search over all pairs also costs $\widetilde{\mathcal{O}}(N)$ gates. A careful analysis shows that, if evaluating the function h costs H gates, Tani's algorithm only provides a $\mathcal{O}(\sqrt{\mathsf{H}})$ advantage over Grover's algorithm [22].

Another approach based on a distributed computing model achieves a very good time-memory tradeoff of $TM = \widetilde{\mathcal{O}}(N)$ [8]. However, this is the wall-clock time of a distributed algorithm, and the gate cost remains $\widetilde{\mathcal{O}}(N)$ at each point of the tradeoff curve. There are also locality issues; achieving this tradeoff requires a *nonlocal* connectivity model or a network that can sort itself in poly-logarithmic time.

The distributed algorithm for multi-target preimage search given in [7] can also be reframed for golden collision search, in which case it becomes a variant of [8] based on iterating a random function and computing "chain-ends" (instead of using a parallel RAM emulation unitary). But it is also inherently parallel and does not reach a smaller gate cost than $\widetilde{\mathcal{O}}(N)$.

Improvements for Specific Quantum Oracles. In this paper, we will consider generic algorithms, and we make no assumption on the function h. In the case of SIKE, Biasse and Pring [10] remarked that a trade-off in quantum search, between the number of iterates and the number of isogenies evaluated, was available. In short, a quantum search with $\mathcal{O}(2^{n/2})$ iterates, each evaluating n isogenies, can be brought down to a cost of $\mathcal{O}(2^{n/2}\sqrt{n}\log_2 n)$ isogeny computations. Thus an advantage similar to Tani's algorithm can be obtained via this modified quantum search.

Random Collision Search. When $h\ : X \to X$ is a random function, a collision can be found in classical time $\mathcal{O}(N^{1/2})$. Brassard *et al.* [12] give a quantum algorithm with time $\widetilde{\mathcal{O}}(N^{1/3})$, using a QRACM of size $\mathcal{O}(N^{1/3})$. In the quantum circuit model, the lowest gate-count to date is obtained with the algorithm of [14]. The algorithm has a gate complexity of $\mathcal{O}(N^{2/5})$ with $\mathcal{O}(N^{1/5})$ *classical memory without random access*, and makes a total of $\mathcal{O}(N^{1/5})$ accesses to the memory.

Quantum Search. Let X be an unstructured search space containing a subset G of good elements. In classical brute-force search, we are given an algorithm Sample_X to sample uniformly at random from X and a function $f\ : X \mapsto \{0,1\}$ such that $f(x) = 1$ if and only if $x \in G$. Then there exists an algorithm Sample_G that samples uniformly from G, which consists in sampling and testing $\mathcal{O}\left(\frac{|X|}{|G|}\right)$ times, until an element of G is found. *Quantum search* uses analogous building blocks and gives a quadratic speedup. Grover's algorithm [19] is the special case of $X = \{0,1\}^n$, which is generalized by Amplitude Amplification [11].

Theorem 2.1 (Adapted from [11], Theorem 2). *Let* $\mathsf{QSample}_X$ *be a quantum circuit that, on input* $|0\rangle$*, produces the uniform superposition over* X*:* $\mathsf{QSample}_X |0\rangle = \frac{1}{\sqrt{|X|}} \sum_{x \in X} |x\rangle$*. Let* O_f *be a circuit that computes* $O_f |x\rangle = (-1)^{f(x)} |x\rangle$*. Then there exists a quantum circuit* $\mathsf{QSample}_G$ *that, on input* $|0\rangle$*, produces* $\frac{1}{\sqrt{|G|}} \sum_{x \in G} |x\rangle$*. It contains* $\mathcal{O}\left(\sqrt{\frac{|X|}{|G|}}\right)$ *calls to* $\mathsf{QSample}_X$ *and* O_f*.*

Thus, we can describe any quantum search (hereinafter a "Grover search") by describing how we compute f and sample from X in superposition.

3 Golden Collision Finding with Random Walks

In this section we define the Magniez, Nayak, Roland, Santha (MNRS¸ [26]) *quantum walk* framework by analogy with classical random walks. We describe Ambainis' algorithm and review van Oorschot-Wiener's golden collision search as a random walk. While this is a needlessly complicated way to describe the classical algorithm, it allows us to quickly introduce a new quantum "iteration-based" walk giving our best G-cost, thanks to the MNRS framework. Next, we give an alternative "prefix-based" walk that reaches the same gate complexity.

3.1 Random Walk Search

A simple, memory-limited search for collisions is to enumerate R random elements of X in a list, sorted by $h(x)$. We find all collisions of h in the list and we check whether any collision is golden. If we do not find any, we delete a random element from the list and replace it with a new random element of X.

To view this as a random walk on a graph, we let the vertices V be the set of all subsets of X of size R. The insertion-and-deletion process moves from one vertex to another. Two vertices are adjacent if and only if they differ in exactly one element. Such a graph is known as a *Johnson graph*, denoted $J(X, R)$.

In general, let $G = (V, E)$ be an undirected, connected, regular graph. We suppose there is some subset of *marked* vertices M, and our task is to output any vertex $x \in M$. We assume we have circuits to perform the following tasks:

Set-up: Returns a random vertex v.
Update: Given a vertex v, returns a random vertex adjacent to v.
Check: Given a vertex v, returns 1 if v is marked and 0 otherwise.

In practice we assume that the random selection is actually performed via a random selection of a bitstring, and a map from bitstrings to the relevant components of the graph; this ensures that the circuits work equally for classically selecting elements at random or for constructing quantum superpositions.

Magniez *et al.* present a unified framework to solve such tasks [26]. The cost depends on several factors:

- The costs S, U, C of the set-up, update, and check circuits, respectively.
- The fraction of marked vertices, $\epsilon := \frac{|M|}{|V|}$.
- The spectral gap of G, denoted δ, equal to the difference between the largest and second-largest eigenvalues of the normalized adjacency matrix of G.

In this paper we only consider Johnson graphs. For a graph $J(X, R)$, S is the cost of initializing a random subset of R elements of X, and U is the cost of replacing one element in such a subset. In all of our applications, it is easiest to keep a single flag bit or counter for the entire list to indicate when it is marked. We update this flag with every update step when we insert and delete elements, and for the check step we simply look at the flag bit.

If we start from a random vertex and take only a few random steps, then the vertex we reach is highly dependent on our starting vertex. For regular graphs, if we take *enough* random steps we reach a uniformly random vertex. The minimum number of steps for this to happen is the *mixing time*, which is the inverse of the spectral gap. For Johnson graphs, it takes R random insertions and deletions to transform one subset of R elements into a new, uniformly random subset. Thus, the mixing time is $\mathcal{O}(R)$ and the spectral gap is $\Omega(1/R)$.

Classical Random Walk. In a classical random walk, we begin by initializing a random vertex with the set-up circuit. We then repeat the following: We take $\mathcal{O}(\frac{1}{\delta})$ random steps in the graph using the update circuit. We then check if the current vertex is marked using the check circuit; if it is marked, we output it and stop, otherwise we repeat the random steps-and-check process. Since $\mathcal{O}(\frac{1}{\delta})$ is the mixing time of the graph, taking this many random steps turns the current vertex into a uniformly random one, which has a ϵ chance of being marked. Thus, the total cost is

$$\mathcal{O}\left(\mathsf{S} + \frac{1}{\epsilon}\left(\frac{1}{\delta}\mathsf{U} + \mathsf{C}\right)\right) \quad . \tag{1}$$

Quantum Random Walk. The quantum walk is analogous to the classical case in the same way that Grover search is analogous to a brute force search. The cost of the quantum random walk is

$$\widetilde{\mathcal{O}}\left(\mathsf{S}+\frac{1}{\sqrt{\epsilon}}\left(\frac{1}{\sqrt{\delta}}\mathsf{U}+\mathsf{C}\right)\right) \quad . \tag{2}$$

If we use the Tolerant Recursive Amplitude Amplification technique from MNRS, possibly using a qubit as control, we can find a marked vertex in $\mathcal{O}(1/\sqrt{\epsilon})$ iterations when ϵ is only a lower bound on the fraction of marked vertices.

In Equation 1, the factor of $\frac{1}{\delta}$ appears because we need that many steps to create a uniformly random vertex. For Johnson graphs, this means we need R insertions and deletions to create a new random list. In the quantum algorithm, Equation 2 seems to imply that we can replace all the elements in an R-element list with only $\sqrt{R}$ insertions and deletions. This is not an accurate description of the quantum walk; to properly describe the algorithm we need to use the graph formalism, but we refer to [26] for full details.

All quantum algorithms in this paper will be quantum walks or quantum walks used as checking oracles in a Grover search. Thus, we will simply describe the graph, the setup, update, and refer to Equation 2. We omit describing the checking subroutine, because in all cases it will simply check if a counter is non-zero or if a flag is 1. The MNRS framework ensures the existence of a corresponding quantum walk and the soundness of our complexity analyses.

To efficiently represent sets for a random walk, a classical computer can use any sorted list structure that enables efficient insertion, deletion and search. For a quantum data structure, we use the Johnson vertex data structure from [22]. In both cases, we can store extra data with each element in the set. This will be necessary for several algorithms.

3.2 Ambainis' Algorithm

Ambainis' element distinctness algorithm [4] performs a random walk and is a query-optimal algorithm for Problem 2.1.

Graph. Ambainis' algorithm uses the Johnson graph $J(X,R)$, where subsets of X are stored as lists of tuples $(x,h(x))$ sorted by $h(x)$, with a a global counter indicating the total number of golden collisions in the current set.

Update. A random step will delete a random element from the set, select a new random element $x\in X$, and insert $(x,h(x))$ into the new list. It must also check if $h(x)=h(y)$ for any y in the list; for all such y, we increment the global counter if the collision is golden. We do the same check for the deleted element, and decrement the counter for any collisions we find.

It costs $\mathsf{H}+\log R$ to compute a new element and insert it into the list, plus the cost to check for golden collisions. The average number of collisions with a new element will be $\frac{R-1}{N}$, since we assume h is a random function. If it costs G to check if a collision is golden, then the total update cost is, on average, $\mathsf{U}=\mathcal{O}(\mathsf{H}+\log R+\frac{R-1}{N}\mathsf{G})$.

Setup. The setup step consists of R insertions into a sorted list, incrementing the counter for any golden collisions we find. This will cost $\mathsf{S} = \mathcal{O}((\mathsf{H}+\log R)R)$.

Marked vertices. A vertex will be marked if it contains both the elements x_g and y_g which form the golden collision. The fraction of such vertices will be $\epsilon = \frac{R(R-1)}{N(N-1)} \approx \frac{R^2}{N^2}$.

Classical Variant. Substituting the previous values into Equation 1, we obtain

$$\mathcal{O}\Bigg(\underbrace{R(\mathsf{H}+\log R)}_{\mathsf{S}} + \underbrace{\frac{N^2}{R^2}}_{1/\epsilon}\Bigg(\underbrace{R}_{1/\delta}\underbrace{\Bigg(\mathsf{H}+\log R+\frac{R-1}{N}\mathsf{G}\Bigg)}_{\mathsf{U}} + \underbrace{1}_{\mathsf{C}}\Bigg)\Bigg). \tag{3}$$

Assuming G is not much more expensive than H, the optimal occurs when $R = \frac{N^2}{R-1}$, and we conclude that $R = N$ is best, with a cost of roughly $\mathcal{O}(N\mathsf{H})$.

Quantum Variant. Assuming cheap QRAQM, the setup, update and checking costs are the same as classically (There are subtle issues ensuring that each subroutine is reversible and constant-time, but we ignore those for now). Equation 2 gives the following complexity:

$$\widetilde{\mathcal{O}}\Bigg(\underbrace{R(\mathsf{H}+\log R)}_{\mathsf{S}} + \underbrace{\sqrt{\frac{N^2}{R^2}}}_{1/\sqrt{\epsilon}}\Bigg(\underbrace{\sqrt{R}}_{1/\sqrt{\delta}}\underbrace{\Bigg(\mathsf{H}+\log R+\frac{R-1}{N}\mathsf{G}\Bigg)}_{\mathsf{U}} + \underbrace{1}_{\mathsf{C}}\Bigg)\Bigg). \tag{4}$$

We optimize this by taking $R = N^{2/3}$, for a total cost of $\widetilde{\mathcal{O}}(N^{2/3})$.

Costing Memory. We need a constant number of memory accesses to insert into the list and to retrieve the collisions in the list to check if they are golden. If each costs $\Theta(R)$ gates, this changes the total cost to

$$\widetilde{\mathcal{O}}\Bigg(\underbrace{R(\mathsf{H}+R)}_{\mathsf{S}} + \underbrace{\sqrt{\frac{N^2}{R^2}}}_{1/\sqrt{\epsilon}}\Bigg(\underbrace{\sqrt{R}}_{1/\sqrt{\delta}}\underbrace{\Bigg(\mathsf{H}+R+\frac{R-1}{N}\mathsf{G}\Bigg)}_{\mathsf{U}} + \underbrace{1}_{\mathsf{C}}\Bigg)\Bigg). \tag{5}$$

Here, the optimal occurs when $R = \mathsf{H}$, for a total cost roughly $\mathcal{O}(N\sqrt{\mathsf{H}})$. Previous work [22] noticed that Grover's algorithm has gate cost of $\mathcal{O}(N\mathsf{H})$, so Tani's algorithm [32] and Ambainis' algorithm [4] provide, in gate cost, an advantage of $\sqrt{\mathsf{H}}$ over Grover's algorithm. This suggests that we should push more of the costs into the function h if we want to beat Grover's algorithm.

3.3 Iteration-based Walk

Here we present van Oorschot-Wiener's golden collision search as a random walk on a Johnson graph, which is equivalent to the original description. This allows us to easily extend to the quantum version, one of our main results, by simply taking square roots of the relevant terms.

The central idea of [30] is to "lift" the function h via *distinguished points*. We select a random subset X_D of size $|X_D| = \theta N$ for some $\theta < 1$, and denote such points as "distinguished". In practice we choose bitstrings with a fixed prefix. From the random function $h : \{0,1\}^n \to \{0,1\}^n$ we construct a random function $h_D : \{0,1\}^n \to X_D$ such that the collisions of h map to collisions of h_D.

To construct h_D, we iterate h. Since h is a pseudo-random function, there is some probability that $h(x) \in X_D$ for every x. We expect to require $1/\theta$ iterations of h before the output is in X_D. Thus, we pick some u greater than $1/\theta$ and define the following function: $h_D(x) = h^m(x)$, where m is the largest $m \leq u$ such that $h^m(x) \in X_D$; if such an m does not exist, we pick a random $y \in X_D$ and set $h_D(x) = y$. If we choose u as a large multiple of $1/\theta$, we expect the case where we do not reach a distinguished point to be exceedingly rare (see Section A in the Appendix). For now, we will simply say that $u \approx 1/\theta$.

Graph. The graph is the same as Ambainis' algorithm, $J(X, R)$. However, each element in the list is stored as $(x, h_D(x), u_x)$, where u_x is such that $h_D(x) = h^{u_x}(x)$. We will not detect all golden collisions, so we use a global flag rather than a counter to track whether the list contains a golden collision. Section A in the Appendix explains how this can be done in a history-independent way.

Update. To insert a new element, we select a random x from X and iterate $h^i(x)$ until either $i \geq u$ or $h^i(x)$ is distinguished. We then write $(x, h^{u_x}(x), u_x)$ into the list, where u_x is the maximum i we found.

To check for the golden collision, we look for all y such that $h_D(y) = h_D(x)$. This implies there is some n, m such that $h^n(x) = h^m(y)$. We want to find this collision and check if it is golden. Assume without loss of generality that $u_x \geq u_y$. Then we set $x' = h^{u_x - u_y}(x)$, repeatedly apply h to x' and y and compare the results. As soon as they are equal, we check if this is the golden collision, and update the flag bit if it is. We then delete one of the previous elements from the list, and do the same check for golden collisions, setting the flag to 0 if the deleted element was part of the golden collision.

It costs $u\mathsf{H} = \mathcal{O}(\mathsf{H}/\theta)$ to compute $h_D(x)$ for a random insertion of x, and it classically costs $\log R$ to insert that element. To maintain the flag indicating if the list contains a trail that leads to the golden collision, we must locate where the underlying collision of h occurs, which takes $u\mathsf{H}$ steps for each collision. The average number of collisions is $(R-1)u^2/N = \mathcal{O}(R/N\theta^2)$, because there are u points on the trail leading to the newly-inserted point, and for each of the $R-1$ existing elements in the list, its value under h has a u/N chance of ending up in the trail of the new point.

Thus, the update cost becomes $\mathsf{U} = \mathcal{O}\left(\frac{\mathsf{H}}{\theta} + \log R + \frac{R}{N\theta^2}\frac{\mathsf{H}}{\theta} + \frac{R}{N\theta^2}\mathsf{G}\right)$. From here on we assume that $\mathsf{G} \ll u\mathsf{H}$, so we ignore the last term.

Setup. The setup is just R sequential insertions, maintaining the flag bit, which costs $\mathsf{S} = \mathcal{O}(R\,(\mathsf{H}n + \log R))$.

Marked Vertices. Section B in the Appendix gives a detailed analysis of the number of marked elements. Roughly speaking, every random function will produce some number of points ("predecessors") z such that $h^k(z) = x_g$ or $h^k(z) = y_g$ for some k. For a vertex to be marked, we must select at least one predecessor for each half of the golden collision among the R random starting points. More predecessors means a higher chance of finding the golden collision, but selecting a random function that gives many predecessors to the golden collision is unlikely.

To find a large number of predecessors, we can select a random function h' and precompose $h \circ h'$ and perform the search on this new function. This acts like a new random function, but preserves the golden collision. Lemma B.2 shows that for a fixed t, the probability that a random function will give at least t predecessors to both halves of the golden collision is $\Theta(1/t)$. From here on, we assume that the golden collision has at least t predecessors, and we will simply repeat the walk with new functions until it works, which will be $\Theta(t)$ times.

Given such a well-behaved function, each random element has a roughly t/N chance of being a predecessor of one half of the golden collision. We need predecessors of both halves, and there are R vertices, so there are $\Omega(\frac{R^2t^2}{N^2})$ marked vertices (Theorem B.1).

Analysis. Assume $\log R \ll \frac{\mathsf{H}}{\epsilon}$, and that H/θ dominates G, then the cost of a single walk, by Equation 1, is:

$$\mathcal{O}\Bigg(\underbrace{R\Big(\mathsf{H}n + \log R\Big)}_{\mathsf{S}} + \underbrace{\frac{N^2}{R^2t^2}}_{1/\epsilon}\Bigg(\underbrace{R}_{1/\delta}\underbrace{\left(\frac{\mathsf{H}}{\theta} + \log R + \frac{(R-1)n^2}{N}\frac{\mathsf{H}}{\theta}\right)}_{\mathsf{U}} + \underbrace{1}_{\mathsf{C}}\Bigg)\Bigg) \tag{6}$$

$$= \mathcal{O}\left(\frac{R\mathsf{H}}{\theta} + \frac{N^2}{Rt^2\theta}\mathsf{H} + \frac{N}{t^2\theta^3}\mathsf{H}\right). \tag{7}$$

We expect to repeat the walk $\Theta(t)$ times with different random functions before we select one that gives the golden collision sufficiently long trails. Thus, the total cost is

$$\mathcal{O}\left(\frac{tR\mathsf{H}}{\theta} + \frac{N^2}{Rt\theta}\mathsf{H} + \frac{N}{t\theta^3}\mathsf{H}\right). \tag{8}$$

The right two terms are largest, so we optimize those first. The optimal will occur when the two sides are equal: $\frac{N^2}{Rt\theta} = \frac{N}{t\theta^3}$, which implies $\theta = \sqrt{R/N}$. The remaining terms balance when $t = \frac{N}{R}$, giving a cost of $\mathcal{O}(\mathsf{H}N^{3/2}/R^{1/2})$, so long as $R \leq N$. This recaptures van Oorschot and Wiener's result, including their heuristic value of the number of function repetitions.

3.4 Quantum Iteration-Based Walk

As with Ambainis' algorithm, we compute the cost in the quantum case by making the following changes: • the cost to access memory is now $\mathcal{O}(R)$, • the $1/\epsilon$ and $1/\delta$ terms in Eq. 1 get square root speed-ups, as in Eq. 2, • the update subroutine must be reversible and constant-time (Section A in the Appendix gives the details of this change), • we perform a Grover search for random functions, and thus only need to repeat the walk $\mathcal{O}(\sqrt{t})$ times.

We will find that the optimal parameters would put $t \geq 1/\theta^2$, which invalidates our arguments from before. If x_g has t predecessors, with high probability we can still expect $\Omega(1/\theta^2)$ predecessors p such that $h^k(p) = x_g$ for $k \leq 1/\theta$ (Theorem B.2). Thus, the fraction of marked vertices will still be $\epsilon = \Omega(\frac{R^2}{N^2\theta^4})$.

This gives a total cost of

$$\widetilde{\mathcal{O}}\left(t^{\frac{1}{2}}\left(\underbrace{\frac{R\mathsf{H}}{\theta}}_{\mathsf{S}} + \underbrace{\frac{N\theta^2}{R}}_{1/\sqrt{\epsilon}}\left(\underbrace{R^{\frac{1}{2}}}_{1/\sqrt{\delta}}\underbrace{\left(\frac{\mathsf{H}}{\theta} + R + \frac{(R-1)\mathsf{H}}{N\theta^3}\right)}_{\mathsf{U}} + \underbrace{1}_{\mathsf{C}}\right)\right)\right) \tag{9}$$

The cost increases with t so we want to take $t = 1/\theta^2$, the minimum before the fraction of marked vertices increases. Optimizing the rest gives $\theta = \mathsf{H}/R$, $R = N^{2/7}\mathsf{H}^{4/7}$, and a total gate cost of $\boxed{\widetilde{\mathcal{O}}\left(N^{6/7}\mathsf{H}^{5/7}\right)}$.

3.5 Prefix-based Walk

In this alternative quantum walk, we use a slightly altered definition of distinguished points: X_D becomes the set of inputs x such that $h(x)$ has a given prefix. Either both halves of the golden collision are distinguished, which happens with probability θ, or none is. By choosing different prefixes, we can easily change the definition of X_D, and after $\frac{1}{\theta}$ trials, or $\frac{1}{\sqrt{\theta}}$ quantum search iterates, we expect the golden collision to be distinguished.

Graph. The graph becomes $J(X_D, R)$. Elements are stored as tuples $(x, h(x))$, sorted by $h(x)$. The list has a global counter of the number of golden collisions.

Update. To insert a new element $(x, h(x))$, we need to sample randomly from X_D. We use Grover's algorithm for a partial pre-image search on h to find x such that $h(x)$ has the correct prefix. Once we find a random element, we check for the golden collision with existing elements and increment the counter, as in Ambainis' algorithm. We do the same procedure when we delete a random element. Since the fraction of distinguished points is θ, the update cost is $\frac{\mathsf{H}}{\sqrt{\theta}} + R$.[4]

Marked Vertices. A vertex is marked if it contains both halves of the golden collision, chosen among the θN distinguished points. With a wrong prefix, no vertices are marked. With the right prefix, vertices are marked with probability $R^2/(\theta^2 N^2)$.

[4] The quantum search in the update unitary cannot be exact, because the exact size of X_D is not known at runtime. The error depends on the difference between $|X_D|$ and θN for the actual good choice of distinguished points. A "hybrid" argument, as in [4], shows that this has no consequence on the walk.

Analysis. With the correct prefix, we find a marked vertex with $\frac{N\theta}{R}$ iterations; with an incorrect prefix, we will never find a marked vertex. Thus, we use the walk as a checking unitary in a Grover search for the correct prefix. From Equation 2, each walk has a cost of

$$\widetilde{\mathcal{O}}\left(\underbrace{R\frac{\mathsf{H}}{\sqrt{\theta}} + R\log R}_{\mathsf{S}} + \underbrace{\frac{N\theta}{R}}_{1/\sqrt{\epsilon}}\left(\underbrace{\sqrt{R}}_{1/\sqrt{\delta}}\underbrace{\left(\frac{\mathsf{H}}{\sqrt{\theta}} + R\right)}_{\mathsf{U}} + \underbrace{1}_{\mathsf{C}}\right)\right) \quad . \tag{10}$$

Optimizing R and θ gives $R = \mathsf{H}/\sqrt{\theta}$. The walk is sound if $N\theta/R \geq 1$ i.e. $N\theta^{3/2} \geq \mathsf{H}$ i.e. $\theta \geq (\mathsf{H}/N)^{2/3}$. Since there are $1/\theta$ possible prefixes, the Grover search must iterate $1/\sqrt{\theta}$ times. The total gate cost, with the Grover search, is:

$$\widetilde{\mathcal{O}}\left(\frac{1}{\sqrt{\theta}}\left(\frac{\mathsf{H}^2}{\theta} + \underbrace{N\theta^{3/4}\sqrt{\mathsf{H}}}_{\text{Walk}}\right)\right) = \widetilde{\mathcal{O}}\left(\mathsf{H}^2\theta^{-3/2} + N\theta^{1/4}\sqrt{\mathsf{H}}\right) \quad . \tag{11}$$

The minimal gate complexity with this method is reached when $\mathsf{H}^2\theta^{-3/2} = N\theta^{1/4}\sqrt{\mathsf{H}}$ i.e. $\theta = N^{-4/7}\mathsf{H}^{6/7}$. At this point we obtain a total gate cost of $\widetilde{\mathcal{O}}(N\theta^{1/4}\sqrt{\mathsf{H}}) = \widetilde{\mathcal{O}}(N^{6/7}\mathsf{H}^{5/7})$ and corresponding memory $R = N^{2/7}\mathsf{H}^{4/7}$, the same result as in Section 3.3.

Classically, the iteration is appealing because the probability of a collision between two trails is much higher than the probability of a collision between two randomly chosen distinguished points. In contrast, a quantum computer can find preimages of distinguished points faster using Grover search, but cannot iterate a function faster than a classical computer. This advantage and disadvantage cancel out, which is why both methods fall on the same complexity.

4 Parallelization

The algorithms of Section 3 optimize only the *gate cost* and benefit from leaving most of the qubits idle for most of the time. Trying to reduce the depth may or may not increase the gate complexity. For example, the depth of the memory access circuit can be brought down easily to $\mathcal{O}(\log R)$. In contrast, reducing the depth of a Grover search by a factor $\sqrt{P}$ multiplies its gate cost by $\sqrt{P}$.

In this section we optimize the gate count under a depth limit. We find that prefix-based walks can maintain an advantage in gate cost over Grover's algorithm. However, by combining prefix methods with the Multi-Grover algorithm of [8], we provide a much better approach to parallelization under very short depth limits. Even with local connectivity in a two-dimensional mesh, this approach can parallelize to depths as low as $\mathcal{O}(N^{1/2})$ without increasing gate cost over $\mathcal{O}(N)$, and to depths as low as $O(N^{1/4})$ with gate cost $\mathcal{O}(N^{3/2}/\mathsf{D})$. We do not analyze parallel iteration-based walks.

In our computational model, we can apply gates freely to as many qubits as we wish, but it is helpful to think of many *parallel processors* that can act on the circuit all at once. We represent this with a parameter P.

There is always a naive strategy of splitting the search space. Each processor would search a disjoint subset of possible inputs; however, since we want a collision, we need to ensure each *pair* of inputs is assigned to one processor. Thus, with P processors, each one must search a space of size $N/P^{1/2}$. We would like to find better methods, if possible.

4.1 Prefix-based Walk

We consider the algorithm of Section 3.5. The setup step can be perfectly parallelized. We do not parallelize the iterations of the walk nor the overall search for prefixes, but instead use our computing power to accelerate the update step. The depth to find an element with a good prefix can be reduced to $\mathsf{H}_D/(\sqrt{\theta}\sqrt{P})$ by parallelizing the Grover search, as long as we have $P \leq R$ and $P \leq \frac{1}{\theta}$. This increases the total gate cost to

$$\widetilde{\mathcal{O}}\left(\frac{1}{\sqrt{\theta}}\left(\underbrace{\frac{R\mathsf{H}_G}{\sqrt{\theta}}+\mathsf{S}_G}_{\mathsf{S}} + \underbrace{\frac{N\theta}{R}}_{1/\sqrt{\epsilon}}\left(\underbrace{\sqrt{R}}_{1/\sqrt{\delta}}\underbrace{\left(\frac{\mathsf{H}_G\sqrt{P}}{\sqrt{\theta}}+R\right)}_{\mathsf{U}} + \underbrace{1}_{\mathsf{C}}\right)\right)\right) \tag{12}$$

where S_G is the gate cost of sorting each vertex, which will depend on the connectivity. Optimizing the gate cost gives $R = \frac{\mathsf{H}_G\sqrt{P}}{\sqrt{\theta}}$. The constraint $P \leq R$ turns into $\sqrt{P} \leq \mathsf{H}_G/\sqrt{\theta}$ which is implied by the condition $P\theta \leq 1$. Even in the local case, where we have $\mathsf{S}_G = R\sqrt{R}$, we find that $\frac{R\mathsf{H}_G}{\sqrt{\theta}}$ dominates the setup cost. By writing an equality between the setup and the walk, we get $\frac{R\mathsf{H}_G}{\sqrt{\theta}} = N\theta\sqrt{R}$. By replacing R in this equation we find $\theta = N^{-4/7}\mathsf{H}_G^{6/7}P^{1/7}$. This gives $R = \mathsf{H}_G^{4/7}N^{2/7}P^{3/7}$. The total gate cost becomes $\boxed{\widetilde{\mathcal{O}}(N^{6/7}\mathsf{H}_G^{5/7}P^{2/7})}$.

The total depth depends on our assumption about locality, because sorting the vertex in the set-up and inserting into the vertex during an update will both depend on the architecture. For both, the depth will be $\mathcal{O}(\log R)$ in a non-local setting but $\mathcal{O}(R^{1/2})$ in the local setting. If we denote this depth as S_D, the total depth of each walk is

$$\widetilde{\mathcal{O}}\left(\frac{\mathsf{H}_D}{\sqrt{\theta}} + \mathsf{S}_D + \frac{N\theta}{R}\sqrt{R}\left(\frac{\mathsf{H}_D}{\sqrt{\theta}\sqrt{P}} + \mathsf{S}_D\right)\right) . \tag{13}$$

As long as $\mathsf{H}_D/\sqrt{\theta P} \geq \mathsf{S}_D$, the depth does not depend on locality; finding distinguished points takes longer than insertion or sorting. We can parallelize up to $\frac{1}{\sqrt{\theta}\sqrt{P}} = \mathsf{S}_D$ which gives $P = \widetilde{\mathcal{O}}(N^{1/2})$ in the non-local setting and $P = \widetilde{\mathcal{O}}(N^{2/11})$ in the local setting.

Beyond this maximum parallelization of the distinguished point search, we can parallelize the search over possible prefixes. In this case the search for the correct prefix is like a normal Grover search, where the oracle is a maximally-parallelized random walk. We can parallelize this way up to $1/\theta$ processors; beyond this, we split the search space.

Table 1: Asymptotic parameters for prefix-based random walks. For readability, H and $\mathcal{O}$ notations are omitted. The line "Any" describes a tradeoff for any $\mathsf{D} \leq N^{6/7}$, until $\mathsf{D} = N^{1/2}$ in the non-local case and $\mathsf{D} = N^{8/11}$ in the local case. "Inner" parallelism is inside a walk. "Outer" parallelism is in the outer Grover iterations. "Memory" is the width of a single walk.

Locality constraint	Depth limit	G-cost	Memory	Parallelism Inner	Parallelism Outer	DW-cost
Any	$\mathsf{D} = N^{\frac{6}{7}}$	$N^{\frac{6}{7}}$	$N^{\frac{2}{7}}$	1	1	$N^{\frac{8}{7}}$
	$N^* \leq \mathsf{D} \leq N^{\frac{6}{7}}$	$N^{\frac{6}{5}}\mathsf{D}^{-\frac{2}{5}}$	$N^{\frac{4}{5}}\mathsf{D}^{-\frac{3}{5}}$	$N^{\frac{6}{5}}\mathsf{D}^{-\frac{7}{5}}$	1	$N^{\frac{4}{5}}\mathsf{D}^{\frac{2}{5}}$
Non-local	$\mathsf{D} = N^{\frac{1}{2}}$	N	$N^{\frac{1}{2}}$	$N^{\frac{1}{2}}$	1	N
	$N^{\frac{1}{4}} \leq \mathsf{D} \leq N^{\frac{1}{2}}$	$N^{\frac{3}{2}}\mathsf{D}^{-1}$	$N^{\frac{1}{2}}$	$N^{\frac{1}{2}}$	$N\mathsf{D}^{-2}$	$N^{\frac{3}{2}}\mathsf{D}^{-1}$
	$\mathsf{D} \leq N^{\frac{1}{4}}$	$N^2\mathsf{D}^{-3}$	D^2	D^2	D^2	$N^2\mathsf{D}^{-3}$
2-dim. neighbors	$\mathsf{D} = N^{\frac{8}{11}}$	$N^{\frac{10}{11}}$	$N^{\frac{4}{11}}$	$N^{\frac{2}{11}}$	1	$N^{\frac{12}{11}}$
	$N^{\frac{5}{11}} \leq \mathsf{D} \leq N^{\frac{8}{11}}$	$N^{\frac{18}{11}}\mathsf{D}^{-1}$	$N^{\frac{4}{11}}$	$N^{\frac{2}{11}}$	$N^{\frac{16}{11}}\mathsf{D}^{-2}$	$N^{\frac{20}{11}}\mathsf{D}^{-1}$
	$\mathsf{D} \leq N^{\frac{5}{11}}$	$N^2\mathsf{D}^{-\frac{9}{5}}$	$\mathsf{D}^{\frac{4}{5}}$	$\mathsf{D}^{\frac{2}{5}}$	$\mathsf{D}^{\frac{6}{5}}$	$N^2\mathsf{D}^{-\frac{7}{5}}$

Grover's algorithm under a depth limit D will cost $\mathcal{O}(N^2/\mathsf{D})$ gates. Table 1 shows that prefix-based walks are exponentially cheaper than Grover's algorithm, even under restrictive depth limits, though the factor is small.

4.2 Multi-grover Search

For even shorter depth constraints, our next algorithm is a prefix-based adaptation of [8]. As in the prefix-based random walk of Section 3.5, we choose an arbitrary prefix and define distinguished points X_D to be those x where $h(x)$ has the fixed prefix. We wrap the entire algorithm in a Grover search for the correct prefix, which will require $\mathcal{O}(1/\sqrt{\theta})$ iterations.

For each prefix, we run a Grover search over lists of distinguished points. To construct each list, we use each one of the P processors to separately run a Grover search for x such that $h(x)$ is a distinguished point. This has cost $\mathcal{O}(\mathsf{H}_G/\sqrt{\theta})$ per processor, so the total gate count is $\mathcal{O}(\mathsf{H}_G P/\sqrt{\theta})$.

The Grover search will produce a random list of P points out of the $N\theta$ distinguished ones, so for the good prefix choice, the probability of containing the golden collision is at least $\frac{\binom{P}{2}(N\theta)^{P-2}}{(N\theta)^P} = \Omega\left(\frac{P^2}{N^2\theta^2}\right)$.

The Grover search on prefixes requires $\mathcal{O}(1/\sqrt{\theta})$ iterations, leading to a total cost of

$$\mathcal{O}\left(\frac{1}{\theta^{1/2}}\frac{N\theta}{P}\left(\frac{\mathsf{H}_G P}{\theta^{1/2}} + S_G\right)\right) = \mathcal{O}\left(N\mathsf{H}_G + \frac{N\theta^{1/2}S_G}{P}\right) \tag{14}$$

The sorting cost S_G is the interesting factor. If S_G/P is small, then the $\mathcal{O}(N\mathsf{H}_G)$ term will be the greatest and lead to a near-perfect parallelization. This

Table 2: Prefix-based Multi-Grover on a local architecture limited to a depth D. The three different parallelization strategies are described in the text.

Parallelization	Depth limits	G-cost	Total hardware	Depth	DW-cost
DP search	$N^{\frac{1}{2}} \leq \mathsf{D}$	N	$N\mathsf{D}^{-1}$	D	N
Prefix search	$N^{\frac{1}{4}} \leq \mathsf{D} \leq N^{\frac{1}{2}}$	$N^{\frac{3}{2}}\mathsf{D}^{-1}$	$N^{\frac{3}{2}}\mathsf{D}^{-2}$	D	$N^{\frac{3}{2}}\mathsf{D}^{-1}$
Split search space	$\mathsf{D} \leq N^{\frac{1}{4}}$	$N^2\mathsf{D}^{-3}$	$N^2\mathsf{D}^{-4}$	D	$N^2\mathsf{D}^{-3}$

is the original result of [8]. Our improvement is that when S_G/P is large, we can adjust θ to compensate. For example, on a two-dimensional mesh, $S_G = \mathcal{O}(P^{3/2})$. In this case we set $\theta = \mathsf{H}_D^2/P$. The depth to construct each list is $\mathcal{O}(\mathsf{H}_D/\theta^{1/2})$ and we denote the depth to sort as S_D, so the total depth is

$$\mathcal{O}\left(\frac{N\theta^{1/2}}{P}\left(\frac{\mathsf{H}_D}{\theta^{1/2}} + S_D\right)\right) = \mathcal{O}\left(\frac{N\mathsf{H}_D}{P} + \frac{N\theta^{1/2}S_D}{P}\right). \tag{15}$$

In the two-dimensional mesh, $S_D = \mathcal{O}(P^{1/2})$, so we find a total depth of $\mathcal{O}(N\mathsf{H}_D/P)$. Thus, this algorithm parallelizes perfectly, even accounting for locality. The maximum parallelization this method can achieve is $P = \mathcal{O}(N^{1/2})$. At this point, each list contains *all* the distinguished points, so the walk provides no advantage. To reach lower depths, we first parallelize the search over prefixes, which can reach a depth of $O(N^{1/4})$. Below this, we split the search space. Table 2 summarizes these results. If we have some architecture where $S_D = o(P^{1/2})$, we can choose $\theta = P/N$ and the asymptotic depth is $\mathcal{O}(N\mathsf{H}_D/P)$ even for large P.

5 Quantum (Parallel) Collision Search

In this section, we study the algorithm of [14] which, in the baseline quantum circuit model, is the only one that achieves a lower gate count than classical for the collision search problem. Here, our goal is to output any collision from a random function $h : \{0,1\}^n \to \{0,1\}^n$ with many expected collisions. We improve the parallelization given in [14] in order to achieve the best gate counts under a depth restriction. This will help us compare our golden collision search algorithms to some NIST security levels.

Algorithm. The algorithm of [14] uses the same definition of distinguished points as in our prefix-based walk. It runs in two phases: first, Grover's algorithm finds M distinguished points. These elements are stored in a *classical* memory with *sequential access*. Second, we search a distinguished point colliding with the memory. Sampling from distinguished points is done with Grover search. Testing membership in the memory is done with a sequential circuit. The gate complexity is: $\mathcal{O}\left(M\frac{\mathsf{H}_G}{\sqrt{\theta}} + \sqrt{\frac{N\theta}{M}}\left(\frac{\mathsf{H}_G}{\sqrt{\theta}} + M\right)\right)$ and the gate count is optimal when $\mathsf{H}_G/\sqrt{\theta} =$

M and $M^2 = M\sqrt{N\theta/M}$ *i.e.* $\theta = M^3/N$ and $M = \mathsf{H}_G^{2/5}N^{1/5}$. Then we have a gate count of $\mathcal{O}(\mathsf{H}_G^{4/5}N^{2/5})$ [14].

Parallelized Algorithm. The authors of [14] considered the first phase to be distributed on many quantum processors, the distinguished points stored in a single classical memory, and the second phase as a distributed Grover search. Using the methods of [8] does better, similar to our Multi-Grover golden collision search. Each processor has a local classical memory of size M/P, where it stores its distinguished points. We do a Grover search over lists of P elements, so in each iteration each processor finds a new distinguished point. To test these new values of h for a collision in the stored data, we use the quantum parallel RAM emulation unitary of [8, Theorem 5]. It emulates in total gate count S_G (and depth S_D) P parallel calls to a RAM of size P. With each call we compare against the *first* distinguished point stored by each processor, then the second, etc. Assuming $P \leq M$, the gate count and depth become:

$$\mathcal{O}\left(M\frac{\mathsf{H}_G}{\sqrt{\theta}}+\sqrt{\frac{N\theta}{MP}}\left(P\frac{\mathsf{H}_G}{\sqrt{\theta}}+\frac{M}{P}S_G\right)\right) \text{ and } \mathcal{O}\left(\frac{M}{P}\frac{\mathsf{H}_D}{\sqrt{\theta}}+\sqrt{\frac{N\theta}{MP}}\left(\frac{\mathsf{H}_D}{\sqrt{\theta}}+\frac{M}{P}S_D\right)\right)$$

We set $\frac{\mathsf{H}_G}{\sqrt{\theta}} = \frac{M}{P^2}S_G$ and $\theta = M^3/(NP)$. We obtain a gate count of $\frac{M^2}{P^2}S_G = \mathsf{H}_G^{4/5}N^{2/5}S_G^{1/5}$, where S_G is a function of P. hence $S_G^{2/5} \leq \mathsf{H}_G^{2/5}N^{1/5}$. The parallelization from [14] occurs in the worst-case scenario $S_G = P^2$.

Depth Optimization. Assuming $\mathsf{H}_G = \mathsf{H}_D = \mathsf{H}$, on a local 2-dimensional grid, $S_G = P^{3/2}$, $S_D = P^{1/2}$ and the depth is $\mathsf{H}_G^{4/5}N^{2/5}P^{-7/10}$. If we optimize the gate count for a given depth D, we get $P = \mathsf{H}_G^{8/7}N^{4/7}\mathsf{D}^{-10/7}$ and a gate count: $\mathsf{D}P = \mathcal{O}(\mathsf{H}_G^{8/7}N^{4/7}\mathsf{D}^{-3/7})$ which is valid as long as $P \leq N^{1/3}\mathsf{H}_G^{2/3}$ i.e. $N^{1/6}\mathsf{H}_G^{1/3} \leq \mathsf{D}$.

Further Parallelization. When $\mathsf{D} = N^{1/6}\mathsf{H}^{1/3}$, there are $P = N^{1/3}\mathsf{H}^{2/3}$ processors which independently search for distinguished points. These points are sorted and a collision is expected to occur. The sorting step of depth $\sqrt{P}$ is balanced with the search step, and the gate count is $\widetilde{\mathcal{O}}(\sqrt{N}\mathsf{H})$. We reach depths below $\widetilde{\mathcal{O}}(N^{1/6})$ by doing this independently in parallel. We set θ such that $\mathsf{H}\theta^{-1/2} = \mathsf{D}$ *i.e.* $\theta = \mathsf{H}^2\mathsf{D}^{-2}$. We set $P = \mathsf{D}^2$. We now have chunks of processors that look for distinguished points (one per processor), sort themselves and look whether a collision has occurred. The probability to get a collision for a chunk is $\frac{\mathsf{D}^2}{\theta N}$, thus we need $\frac{\theta N}{\mathsf{D}^2} = \frac{\mathsf{H}^2 N}{\mathsf{D}^4}$ chunks. The total gate count is: $\widetilde{\mathcal{O}}\left(\frac{\mathsf{H}^2 N}{\mathsf{D}^4}\mathsf{D}\right) = \widetilde{\mathcal{O}}\left(\frac{\mathsf{H}^2 N}{\mathsf{D}^3}\right)$.

6 Security of SIKE

Supersingular Isogeny Key Encapsulation (SIKE) [20] is a candidate post-quantum key encapsulation based on isogenies of elliptic curves. So far its security is based

Table 3: Security thresholds from NIST. AES key search figures are from [21]. For AES key search, the width is approximately equal to $\max(13, DW - \text{Maxdepth})$. The cost of evaluating SHA-3 is taken from [5].

		AES key search			SHA Collisions			
		Security Level			Security Level			
		1	3	5	2		4	
Metric	Maxdepth				Cost	Width	Cost	Width
G-cost	∞	83	116	148	122	12	184	17
	2^{96}	83	126	191	134	50	221	143
	2^{64}	93	157	222	148	96	268	221
	2^{40}	117	181	246	187	158	340	317
DW-cost	∞	87	119	152	134	12	201	17
	2^{96}	87	130	194	145	50	239	143
	2^{64}	97	161	225	159	96	285	221
	2^{40}	121	185	249	198	158	357	317
Classical		143	207	272	146	–	210	–

on the hardness of generic meet-in-the-middle attacks. SIKE is parameterized by the bit-length of a public prime parameter p (so SIKE-434 uses a 434-bit prime). The meet-in-the-middle attack must search a space of size $\mathcal{O}(p^{1/4})$. Thus, replacing N with $p^{1/4}$ in our algorithms gives the performance against SIKE.

NIST defined security levels relatively to quantum generic attacks on symmetric primitives. Levels 1, 3, and 5 are defined relatively to an exhaustive key search on the AES block cipher. NIST used gate counts from [18], but we use improved numbers from [21]. They are given in Table 3. Levels 2 and 4 are based on searching for collisions for the SHA family of hash functions. We use the collision search of [14] and the results of Section 5. Table 3 shows the resulting costs when applied to SHA3 under NIST's depth restrictions. SIKE-434, SIKE-503, SIKE-610, and SIKE-751 target NIST's security levels 1, 2, 3, and 5, respectively.

NIST restricts the total circuit depth available by a parameter "Maxdepth". Quantum search algorithms parallelize very poorly so a depth limit forces enormous hardware requirements.

Security estimates. Because of the depth restriction, we focus on the parallel prefix-based walk and parallel Multi-Grover. Overall our results are likely to underestimate the real cost by constant or poly-logarithmic factors. For example, the depth of a 2-dimensional mesh sorting network of R elements is not exactly $R^{1/2}$, but likely closer to $3R^{1/2}$ [23]. We also need estimates of the cost of H, and we use those from [22]. Table 4 shows the costs to attack various SIKE parameters[5] under different depth restrictions, and shows by how many bits

[5] At the moment, we have not tried to combine our results with the technique of [10], which can reduce the oracle's footprint in the case of SIKE. We reckon that their

the attacks exceed the cost thresholds for the NIST security levels. The attacks are parallelized only as much as necessary, using the methods from Section 4. Overall, we find that our attacks lower the quantum security of SIKE compared to the results of [22], but not enough to reduce the claimed security levels.

The asymptotically improved gate cost of the prefix-based walk is barely noticeable because of the depth restrictions. There is a stark difference between the gate cost and the depth×width cost, but only with unrestricted depth. Multi-Grover outperforms the prefix-based walk in nearly all contexts, even in gate cost, because of its parallelization.

On a non-local architecture, the Multi-Grover algorithm parallelizes almost perfectly. The lowest gate costs in Table 4 would apply at all maximum depth values, complicating the security analysis: SIKE-610 would not reach level 3 security in G-cost under a depth limit of 2^{40}, but would reach level 3 at higher depth limits or in DW-cost; SIKE-751 would only reach level 5 security with a depth limit of 2^{96}. Thus, the security level of SIKE depends on one's assumptions about plausible physical layouts of quantum computers. However, the margins are relatively close, and more pessimistic evaluations of the quantum costs of isogeny computations (the factor H) could easily bring SIKE-610 and SIKE-751 back to their claimed security levels, even with a non-local architecture.

7 Conclusion

In this paper, we gave new algorithms for *golden collision* search in the quantum circuit model. We improved the gate counts and depth-width products over previous algorithms when cheap "qRAM" operations are not available. In this model, the NIST candidate SIKE offers less security than claimed in [22], but still more than the initial levels given in [20].

Using two different techniques, we arrived at a gate complexity of $\widetilde{\mathcal{O}}(N^{6/7})$ for golden collision search. The corresponding memory used is $N^{2/7}$. Interestingly, our algorithms actually achieve the same tradeoff between gate count T and quantum memory R as the previous result of Ambainis [4]: $T^2 \times R = N^2$. On the positive side, this shows that qRAM is not necessary if we use less than $N^{2/7}$ memory.

Acknowledgments. A.S. would like to thank André Chailloux and María Naya-Plasencia for helpful discussions. This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement no. 714294 - acronym QUASYModo). S.J. was supported by the University of Oxford Clarendon fund, and would like to thank Christophe Petit and Richard Meister for helpful comments. Both authors would like to thank Steven Galbraith for helpful comments.

tradeoff will bring a small improvement of the numbers in Table 4, both for quantum walks and Multi-Grover.

Table 4: Costs of quantum attacks on SIKE. A non-local Multi-Grover attack would have the same cost at all depth limits presented, equal to the values in the first row. The best value for a given metric and depth constraint is in **bold**. We give a comparison with [22] (Grover, Tani and vOW's algorithms) and [10] (improved oracle in Grover search), though neither of these account for locality. Code to produce these estimates available at `https://project.inria.fr/quasymodo/golden-collision-costs-tar/`.

		Local Prefix-based walk				Local Multi-Grover				Previous [22,10]	
		SIKE p bitlength				SIKE p bitlength				SIKE p length	
Metric	Depth	434	503	610	751	434	503	610	751	434	610
G	∞	**109**	**124**	**147**	**178**	130	148	175	211	124 [22]	169 [22]
	2^{96}	**110**	**134**	184	**255**	130	148	**182**	**255**	143 [22]	200 [22]
	2^{64}	**145**	**181**	235	**307**	154	189	243	314	**145** [22]	**189** [22]
	2^{40}	**184**	**219**	**274**	**345**	186	221	275	346		
DW	∞	150	170	202	243	130	**148**	175	**211**	**126** [10]	**170** [10]
	2^{96}	149	170	223	294	**131**	**158**	**192**	**265**	157 [22]	248 [22]
	2^{64}	174	209	264	336	163	**198**	**252**	**322**	**145** [22]	289 [22]
	2^{40}	205	241	296	367	**187**	**222**	**276**	**346**		
Width	∞	51	57	65	76	10	10	10	11	10 [10]	10 [10]
	2^{96}	53	74	127	199	35	63	96	170	62 [22]	115 [22]
	2^{64}	110	146	200	272	99	134	188	258	91 [22]	136 [22]
	2^{40}	166	201	256	328	147	182	236	306		

References

1. Aaronson, S., Shi, Y.: Quantum lower bounds for the collision and the element distinctness problems. J. ACM 51(4), 595–605 (Jul 2004)
2. Adj, G., Cervantes-Vázquez, D., Chi-Domínguez, J.J., Menezes, A., Rodríguez-Henríquez, F.: On the cost of computing isogenies between supersingular elliptic curves. In: SAC 2018. pp. 322–343. LNCS 11349
3. Albrecht, M., Player, R., Scott, S.: On the concrete hardness of Learning with Errors. Journal of Mathematical Cryptology pp. 169–203 (2015)
4. Ambainis, A.: Quantum walk algorithm for element distinctness. SIAM J. Computing 37, 210–239 (2007)
5. Amy, M., Matteo, O.D., Gheorghiu, V., Mosca, M., Parent, A., Schanck, J.M.: Estimating the cost of generic quantum pre-image attacks on SHA-2 and SHA-3. In: SAC 2016. pp. 317–337 (2016)
6. Arunachalam, S., Gheorghiu, V., Jochym-O'Connor, T., Mosca, M., Srinivasan, P.V.: On the robustness of bucket brigade quantum ram. New Journal of Physics 17(12), 123010 (2015)
7. Banegas, G., Bernstein, D.J.: Low-communication parallel quantum multi-target preimage search. In: SAC. LNCS, vol. 10719, pp. 325–335. Springer (2017)
8. Beals, R., Brierley, S., Gray, O., Harrow, A.W., Kutin, S., Linden, N., Shepherd, D., Stather, M.: Efficient distributed quantum computing. Proc. Royal Soc. London A: Mathematical, Physical and Engineering Sciences 469 (2013)
9. Bennett, C.H.: Time/space trade-offs for reversible computation. SIAM J. Comput. 18(4), 766–776 (1989)

10. Biasse, J.F., Pring, B.: A framework for reducing the overhead of the quantum oracle for use with grover's algorithm with applications to cryptanalysis of sike. J. Math. Cryptol. (2019)
11. Brassard, G., Hoyer, P., Mosca, M., Tapp, A.: Quantum amplitude amplification and estimation. Contemporary Mathematics 305, 53–74 (2002)
12. Brassard, G., Høyer, P., Tapp, A.: Quantum cryptanalysis of hash and claw-free functions. In: LATIN. LNCS, vol. 1380, pp. 163–169. Springer (1998)
13. Buhrman, H., Dürr, C., Heiligman, M., Høyer, P., Magniez, F., Santha, M., de Wolf, R.: Quantum algorithms for element distinctness. SIAM J. Comput. 34(6), 1324–1330 (2005)
14. Chailloux, A., Naya-Plasencia, M., Schrottenloher, A.: An efficient quantum collision search algorithm and implications on symmetric cryptography. In: ASIACRYPT (2). LNCS, vol. 10625, pp. 211–240. Springer (2017)
15. Costello, C., Longa, P., Naehrig, M., Renes, J., Virdia, F.: Improved classical cryptanalysis of SIKE in practice. In: PKC 2020. LNCS 12111
16. Flajolet, P., Odlyzko, A.M.: Random mapping statistics. In: EUROCRYPT. LNCS, vol. 434, pp. 329–354. Springer (1989)
17. Giovannetti, V., Lloyd, S., Maccone, L.: Architectures for a quantum random access memory. Physical Review A 78(5), 052310 (2008)
18. Grassl, M., Langenberg, B., Roetteler, M., Steinwandt, R.: Applying grover's algorithm to AES: quantum resource estimates. In: PQCrypto. Lecture Notes in Computer Science, vol. 9606, pp. 29–43. Springer (2016)
19. Grover, L.K.: A Fast Quantum Mechanical Algorithm for Database Search. In: Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing 1996. pp. 212–219. ACM (1996)
20. Jao, D., Azarderakhsh, R., Campagna, M., Costello, C., Feo, L.D., Hess, B., Jalali, A., Koziel, B., LaMacchia, B., Longa, P., Naehrig, M., Renes, J., Soukharev, V., Urbanik, D.: Supersingular isogeny key encapsulation. Submission to NIST post-quantum project (November 2017), `https://sike.org/#nist-submission`
21. Jaques, S., Naehrig, M., Roetteler, M., Virdia, F.: Implementing grover oracles for quantum key search on AES and LowMC. In: EUROCRYPT 2020. LNCS 12106 (2020)
22. Jaques, S., Schanck, J.M.: Quantum cryptanalysis in the RAM model: Claw-finding attacks on SIKE. In: CRYPTO 2019. pp. 32–61. LNCS 11693 (2019)
23. Kunde, M.: Lower bounds for sorting on mesh-connected architectures. Acta Inf. 24(2), 121–130 (1987)
24. Kuperberg, G.: Another subexponential-time quantum algorithm for the dihedral hidden subgroup problem. In: TQC 2013. pp. 20–34. LIPIcs 22 (2013)
25. Levin, R.Y., Sherman, A.T.: A note on Bennett's time-space tradeoff for reversible computation. SIAM J. Comput. 19(4), 673–677 (1990)
26. Magniez, F., Nayak, A., Roland, J., Santha, M.: Search via quantum walk. SIAM Journal on Computing 40, 142–164 (2011)
27. Naya-Plasencia, M., Schrottenloher, A.: Optimal Merging in Quantum k-xor and k-sum Algorithms. In: EUROCRYPT 2020. LNCS 12106 (2020)
28. Nielsen, M.A., Chuang, I.: Quantum computation and quantum information. AAPT (2002)
29. NIST: Submission requirements and evaluation criteria for the post-quantum cryptography standardization process (2016), `https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/call-for-proposals-final-dec-2016.pdf`

30. van Oorschot, P., Wiener, M.: Parallel collision search with cryptanalytic applications. J.Cryptology 12(1), 1–28 (Jan 1999)
31. Rényi, A., Szekeres, G.: On the height of trees. Journal of the Australian Mathematical Society 7(4), 497–507 (Nov 1967)
32. Tani, S.: An improved claw finding algorithm using quantum walk. In: MFCS. LNCS, vol. 4708, pp. 536–547. Springer (2007)

A Quantum Circuits for Iterations

This section details the quantum circuits used in the quantum iteration-based walk of Section 3.4. The MNRS framework describes the circuit for a quantum random walk, given circuits for the set-up, update, and check subroutines. Beacuse the set-up can done with sequential insertion steps (which are part of the update), and the check step only considers a single cunter or flag, the main analysis is the update step. We use the Johnson vertex data structure from [22]. This is sufficient to describe the steps for the prefix-based walk, but the iteration-based walk is more complicated.

The update will need to do the following:

1. Select a new point in superposition, and iterate the function h until it finds a distinguished point.
2. Find any collisions of the new distinguished point in the existing list.
3. Retrace the trails of any distinguished point collisions to find the underlying collisions of h.

A.1 Iterating the function

Given a randomly selected point x, we define the *trail* of x to be the sequence $(x, h(x), h^2(x), \ldots, h^{n_x}(x))$, where $h^{n_x}(x)$ is distinguished. The goal of this subcircuit is to map states $|x\rangle$ to $|x\rangle\,|h^{n_x}(x)\rangle\,|n_x\rangle$. Unlike classical distinguished-point finding, the quantum circuit cannot stop when it reaches a distinguished point. Rather, we must preselect a fixed number of iterations which will almost certainly reach a distinguished point.

The length of trails is geometrically distributed [30], with a mean equal to $1/\theta$ if the fraction of distinguished points is θ. Using n iterations, the proportion of trails with length greater than $n = c/\theta$ is approximately e^{-c} [30].

Pebbling. Since h is by definition non-injective, it cannot be applied in-place, so we will need a pebbling strategy (see *e.g.* [7,9,25]). We can choose a simple strategy with $2\sqrt{u}$ qubit registers that we will call "baby-step giant-step". We assume u is a perfect square for ease of description. One iteration of h is a "baby step", and a "giant step" is $\sqrt{u}$ iterations. To compute a giant step, we compute $\sqrt{u}$ sequential baby steps with no uncomputation, then uncompute all but the last. Thus, it takes $2\sqrt{u}$ iterations and $\sqrt{u}+1$ registers to take one giant step.

It takes $\sqrt{u}$ giant steps to reach $h^u(x)$, and we will keep each giant step until the end before uncomputing. Thus, the total cost is $4u$ sequential iterations of h, and we need $2\sqrt{u}$ registers.

Output. To output the last distinguished point that h reaches, we have a list of k potential distinguished points, all initialized to $|0\rangle$. At every iteration of h, we perform two operations, controlled on whether the new output is distinguished. The first operation cycles the elements in the list: the ith element is moved to location $i + 1 \mod k$. Then the output is copied to the first element in the list.

As long as the iterations reach less thank k distinguished points in *total*, this will put the last distinguished point at the front of the list, where we can copy it out. If there are more than k points reached, the copy operation, consisting of CNOT gates, will produce the bitwise XOR of the new and old distinguished points in the list. This will not cause issues in the random walk, but it is highly unlikely to detect a collision. Thus, we can regard this as reducing the number of marked vertices. By Markov's inequality the probability of more than k points is at most $\frac{c}{k}$, and even smaller if we assume a binomial distribution of the number of distinguished points in a trail.

Error Analysis. Errors occur if a trail finds zero or too many distinguished points. F The only points we need to operate correctly are those leading to the golden collision. Starting from a vertex that would be marked if we had a perfect iteration circuit, it contains two elements that lead to the golden collision (see Section B). If either element produces an incorrect iteration output, the circuit will incorrectly conclude that the vertex is not marked[6]. Suppose that some number of points t will produce trails that meet at the golden collision. In the worst case, the probabilities of failure for each point are dependent (say, some point on the trail just before the golden collision causes the error). Then there will be a probability of roughly p that the entire algorithm fails, and a probability roughly $1 - p$ that it works exactly as expected. In this case, we will need to repeat the walk with another random function.

For $p \in \Omega(1)$, such imperfections add only an $\mathcal{O}(1)$ cost to the entire algorithm. Thus, based on the previous analyses, we can choose u to be a small, constant multiple of $1/\theta$, and choose k to be a constant as well.

Locality. The iteration can be done locally in many ways. For our baby-step giant-step pebbling, we can arrange the memory into two loops so that the giant steps are stored in one loop and baby steps in the other. We can then sequentially and locally compute all the baby steps, and ensure that the final register is close to the starting register. Then we can copy the output – which is a giant step – into the loop for giant steps. Then we cyclically shift all the giant steps, which is again local. These loops do not change the time complexity at all, are easy to create in a two-dimensional nearest-neighbour architecture.

Thus, our algorithms retain their gate complexity in a two-dimensional nearest-neighbour architecture, and have a time complexity asymptotically equal to their gate complexity in this model.

[6] If both produce incorrect outputs we may find the marked vertex if they produce the same incorrect value, but the probability of this is vanishingly small.

A.2 Finding Collisions

According to the optimizations in Section 3.4, the average number of collisions per inserted point is $\frac{(R-1)u^2}{N}$ and we choose $R \approx u \approx N^{2/7}$; thus, we have a vanishing expected number of collisions.

This makes our collision-finding circuit simple. We can slightly modify the search circuit on a Johnson vertex [22]. That search circuit assumes a single match to the search string, and so it uses a tree of CNOT gates to copy out the result. With multiple matches, it would return the XOR of all matches. To fix this, we use a constant number t of parallel trees, ordered from 1 to t, and add a flag bit to every node.

Our circuit will first fan out the search string to all data in the Johnson vertex, copy out any that match to the leaf layer of the first tree, and flip the flag bit on all matches. Then it will copy the elements up in a tree; however, it will use the flag bit to control the copying. When copying from two adjacent elements in tree i, one can be identified as the "first" element (perhaps by physical arrangement). If both flag bits are 1, we copy the second element to the first tree where the flag bit for that node is 0, then copy the first element to the higher layer. In any other case, we CNOT each node to its parent. The root nodes of all the trees will be in some designated location, and we can process them from there.

Such a circuit with t trees will correctly copy out any number of collisions up to t. If there are more collisions, it will miss some: they will not be copied out to another tree, and so they will be lost.

A.3 Finding Underlying Collisions

Here we describe how to detect, given two elements (x, n_x) and (y, n_y) with $h^{n_x}(x) = h^{n_y}(y)$, whether they reach the golden collision.

We initialize a new register r_n containing n, the maximum path length from the iteration step. We then iterate h simultaneously for x and y, using the same pebbling strategy as before. We make one small change: At each step we compare r_n to n_x and n_y. If $r_n \leq n_x$, then we apply h to the current x output, and otherwise we just copy the current x output. We do the same for y. This ensures that at the ith step, both trails are $n - i$ steps away from the common distinguished point, so they will reach the collision at the same time.

After each iteration, we apply the circuit to test if a collision is golden, controlled on whether the current output values for x and y are equal. If the collision is golden, we flip an output bit.

A.4 Detecting Marked Vertices

After the circuit in Section A.2, we have a newly-inserted point x, its output $h^{n_x}(x)$ and n_x, as well as (up to) t candidate collisions $y_1, \ldots, y_t$ and their associated numbers n_{y_i}. Our goal is to decide whether the vertex is now marked.

A naive search for the golden collision among each candidate collision will introduce a history dependence. For example, if we insert the golden collision with no extraneous collisions, we will detect it and flip a flag for the vertex. If we then insert more than t predecessors of one half of the golden collision, then we might remove the other half of the golden collision but not detect it, because it might not appear in the list of t candidate collisions.

To avoid this, we modify the circuit based on the number of candidate collisions. If there is exactly one candidate collision, we check for a golden collision with the new point and the candidate collision. If there are are more than two candidate collisions then we do not do any check at all. If it has exactly two candidate collisions, we check for the golden collision between the two candidate collisions (*i.e.* those already in the list).

Theorem A.1 ensures the marked vertices will be precisely those with *exactly* one predecessor from each half of the golden collision. In Section B we find that this has negligible impact on the cost; the probability of choosing a predecessor of the golden collision is so small that there are only a tiny handful of vertices which have more than 2 predecessors, and so we can safely ignore them.

Theorem A.1. *Using the circuit above with $t \geq 3$ ensures that a vertex is marked if and only if it contains exactly 1 predecessor for each half of the golden collision.*

Proof. Suppose every vertex is correctly marked in this way. We will show that one update maintains this property.

If the vertex has no predecessors of the golden collision, then a newly inserted element will not create a collision, and the vertex will not become marked.

If the vertex has exactly one predecessor of the golden collision, then it will not be marked. If a newly inserted element forms a collision with this predecessor, then we run the golden collision detection circuit. If the new point is a predecessor of the same half, the vertex remains unmarked; if it is a predecessor of the other half, the new vertex becomes marked.

If the vertex has two predecessors of one half of the golden collision, then when a new element is inserted that collides with these, we run a circuit that only checks for a golden collision among the existing two predecessors. It will not find the golden collision, so it will not flip the "marked" flag for the vertex, so the vertex remains unmarked. This is correct, since the updated vertex will have more than 1 predecessor for one half of the golden collision.

If the vertex has exactly 1 predecessor for each half, it starts marked. When a new element is inserted, we run a circuit that looks for the golden collision among the existing collisions. This circuit will find a collision, and flip the "marked" flag, which un-marks the vertex. The vertex now contains 2 predecessors for one half of the golden collision, so this is correct.

The vertex has more than two predecessors of the golden collision if and only if the circuit detects more than two collisions. In this case, the vertex will not be marked, and we will not run either detection circuit, so it remains unmarked. □

Multiple golden collisions. If there are multiple golden collisions, the previous method functions almost correctly. If a vertex contains more than one golden collision, there may be some history dependence if one is a predecessor of the other. We can regard this as an imperfect update. The error is at most ϵ^2, and since we only iterate $1/\sqrt{\epsilon\delta}$ walk steps, this causes no problems.

Errors in Random Walks. We will encounter two types of error for the update procedure U. In Section 3.4, we have false negatives: the update will sometimes incorrectly miss a marked vertex, but it will never incorrectly identify an unmarked vertex as marked. Furthermore, these errors are not history-dependent. Thus, we can redefine the underlying set of marked vertices to be precisely the vertices that are correctly identified. This switches our perspective from an imperfect circuit on a perfect graph, to a perfect circuit for an imperfect graph.

If the fraction of marked vertices changes from ϵ to ϵ', then the total runtime changes from

$$\mathcal{O}\left(\mathsf{S}+\frac{1}{\sqrt{\epsilon}}\left(\frac{1}{\sqrt{\delta}}\mathsf{U}+\mathsf{C}\right)\right) \text{ to } \mathcal{O}\left(\mathsf{S}+\frac{1}{\sqrt{\epsilon'}}\left(\frac{1}{\sqrt{\delta}}\mathsf{U}+\mathsf{C}\right)\right) \tag{16}$$

and thus the change in cost is at most a factor of $\mathcal{O}(\sqrt{\epsilon/\epsilon'})$. This means any $\Omega(1)$ reduction in the fraction of marked vertices will incur only a $\mathcal{O}(1)$ increase in the cost of the walk.

In Section 3.5, the update contains a Grover search, which is not exact. This means the actual update circuit U' is close to U, but with some error amplitude, independent of the vertex. This error can be exponentially reduced with more Grover iterations so that after an exponential number of updates, the total error amplitude (and the probability of success of the algorithm) remains constant.

B Probability Analysis

The analysis of van Oorschot and Wiener [30] rests on several heuristic assumptions and numerical evidence for those assumptions. Since we analyze their algorithm as a random walk, these heuristics do not help our analysis. Thus, we must explicitly prove several results about random functions for our algorithm (see [16] for other standard results).

We define the set of predecessors of x as $\mathcal{P}_x = \{y \in X\,|h^n(y) = x, n \geq 0\}$.

We then let $P_x = |\mathcal{P}_x|$. Our goal is to provide distributions of both the number of predecessors, the total height of the tree of predecessors, and the joint distribution among both halves of a particular collision.

Lemma B.1. *The probability that a random function $h : X \to X$ is chosen such that $P_x = t$ is given by*

$$\Pr[P_x = t] = \frac{t^{t-1}}{e^t t!}\left(1+\mathcal{O}(\tfrac{1}{N})\right) \tag{17}$$

for $t = o(N)$. In particular, $\Pr[P_x \geq t] = \Theta(1/\sqrt{t})$.

Proof. We count the number of such functions. To form x's predecessors, we select $t-1$ elements out of the $N-1$ elements which are not x. These form a tree with x as the root. There are t^{t-2} undirected trees (Cayley's formula), which then uniquely defines a direction for each edge to put x at the root. Then the remaining $N-t$ points must map only to themselves. There are $(N-t)^{N-t}$ ways to do this. Then we have N choices for the value of $h(x)$. There are N^N random functions total, giving a probability of

$$\frac{\binom{N-1}{t-1}t^{t-2}N(N-t)^{N-t}}{N^N} = \frac{t^{t-1}}{t!}\frac{N!}{N^N}\frac{(N-t)^{N-t}}{(N-t)!}. \tag{18}$$

Stirling's formula, applied to terms with N, gives an approximation of

$$\frac{t^{t-1}e^{-t}}{t!}\sqrt{\frac{N}{N-t}}\left(1+\mathcal{O}(\tfrac{1}{N})\right). \tag{19}$$

Since $\frac{N}{N-t} = 1 + \frac{t}{N-t} = 1 + \mathcal{O}(1/N)$, we get the first result. For the second, we use Stirling's approximation again to show that $\Pr[P_x = t] \sim \frac{1}{\sqrt{2\pi t^3}}$. An integral approximation gives the asymptotics. □

Lemma B.2. *Fix $x, y \in X$. Let h be a random function under the restriction that $h(x) = h(y)$. Then for $t, s = o(N)$,*

$$\Pr[P_x = t, P_y = s] = \frac{t^{t-1}s^{s-1}}{e^t t! e^s s!}\left(1+\mathcal{O}(\tfrac{1}{N})\right) \tag{20}$$

and the probability that x and y both have at least t predecessors is $\Theta(1/t)$.

Proof. First, x and y can only have the same set of predecessors if they are in the same cycle, but they cannot be in the same cycle because $h(x) = h(y)$. Thus either their sets of predecessors are disjoint, or x is a predecessor of y (meaning $h(x)$ is a predecessor of y). We assume $s \geq t$ without loss of generality, meaning y cannot be a predecessor of x.

When the sets of predecessors are disjoint, we select $t-1$ elements to be predecessors of x from the $N-2$ elements that are neither x nor y, then $s-1$ elements out of the remainder to be predecessors of y. Then we map the remaining elements to themselves, then pick one of the $N-t-s$ elements that are not predecessors of x or y to be the element $h(x)$. The probablity of such a function is

$$\frac{\binom{N-2}{t-1}t^{t-2}\binom{N-t-1}{s-1}s^{s-2}(N-t-s)^{N-t-s}(N-t-s)}{N^{N-1}}. \tag{21}$$

This can be simplified and then approximated to

$$\frac{t^{t-1}}{t!}\frac{s^{s-1}}{s!}\frac{N!}{N^N}\frac{(N-t-s)^{N-t-s}}{(N-t-s)!}\frac{N-t-s}{N-1} = \frac{t^{t-1}}{e^t t!}\frac{s^{s-1}}{e^s s!}(1+\mathcal{O}(\tfrac{1}{N})). \tag{22}$$

Our goal is now to show that the remaining term, where x is a predecessor of y, is of order $\mathcal{O}(1/N)$.

If x is a predecessor of y, we choose $s-2$ predecessors of y (one will be x), and of those, we choose $t-1$ to be predecessors of x. Then we form a tree behind x, then we form a tree of the remaining $s-t$ elements. Then we must attach the two trees: There are $s-t$ choices for where to attach x, i.e., $s-t$ choices for $h(x)$. This forces $h(y)$ to a specific value. From there, the remaining $N-s$ non-predecessor elements map to themselves. The probability of this type of function is

$$\frac{\binom{N-2}{s-2}\binom{s-2}{t-1}t^{t-2}(s-t)^{s-t-2}(s-t)(N-s)^{N-s}}{N^{N-1}}. \tag{23}$$

This can be simplified to

$$\frac{t^{t-1}}{t!}\frac{(s-t)^{s-t}}{(s-t)!}\frac{N!}{N^N}\frac{(N-s)^{N-s}}{(N-s)!}\frac{1}{N-1} \tag{24}$$

which, up to errors of order $\mathcal{O}(1/N)$, equals

$$\frac{1}{N}\left(\frac{t^{t-1}}{e^t t!}\frac{(s-t)^{s-t}}{e^{s-t}(s-t)!}\right) \tag{25}$$

which fits within the error term of Equation 22, since $s-t\leq s$. □

Lemma B.3. *Let n_x be the height of the predecessors of x: the largest integer such that there is some $p\in X$ with $h^{n_x}(p)=x$. Define n_y similarly. Suppose x has t predecessors and y has s predecessors. For $c>0$, the probability that $n_x>c\sqrt{2\pi t}$ or $n_y>c\sqrt{2\pi s}$ is at most*

$$\frac{2(\pi-3)}{3(c-1)^2}\left(1+\mathcal{O}(\tfrac{s}{N})\right). \tag{26}$$

Proof. We can assume that x and y have disjoint trees of predecessors; the case where one is a predecessor of the other fits in the $\mathcal{O}(\frac{s}{N})$ error term.

By [31], the height of a random tree on t vertices has expected value $\sqrt{2\pi t}$ with variance $\frac{2\pi(\pi-3)t}{3}$. Chebyshev's equality implies that the probability that $n_x>c\sqrt{2\pi t}$ is at most $\frac{\pi-3}{3(c-1)^2}$, and this is the same probability that $n_y>c\sqrt{2\pi s}$. The union bound gives the main term of the result.

If x is part of a cycle, then n_x is infinite. This can only occur if $h(x)$ is a predecessor of x, which occurs with probability t/N, hence the error term, which also accounts for infinite n_y. □

We now conclude how many vertices will be marked, assuming that x and y have predecessors with small height. A vertex is marked if and only if it contains exactly one predecessor of x and one predecessor of y.

Theorem B.1. *Let h be a function such that $h(x)=h(y)$, x has t predecessors and the largest trail leading to x has $n_x\leq u$ points, y has s predecessors and the largest trail leading to y has $n_y\leq u$ points. Then the fraction of marked vertices in the graph defined in Section 3.3 (with u iterations of h for each point) is*

$$\Theta\left(\frac{R^2ts}{N^2}\right). \tag{27}$$

Proof. Define the u-predecessors of x by

$$\mathcal{P}_u(x) = \{p \in X \,|h^m(p) = x, u \geq m \geq 0\}\,. \tag{28}$$

A vertex is defined by R random distinct points from X. It will be marked if and only if it contains exactly one point from $\mathcal{P}_u(x)$ and exactly one from $\mathcal{P}_u(y)$. Since $n_x, n_y \leq u$, the sizes of these sets are t and s. This acts as a multinomial distribution, and thus the probability of one element from each set is

$$\binom{R}{2}\frac{t}{N}\frac{s}{N}\left(1 - \frac{t+s}{N}\right)^{R-2} = \Theta\left(\frac{R^2ts}{N^2}\right). \tag{29}$$

□

This covers the case where h has given the golden collision few predecessors, but we may also wish to analyze functions that give more predecessors. We expect this to increase the odds of detecting the golden collision, since there will probably be more close predecessors, even though the height of the predecessors will be large. However, it is sufficient for us to prove that, with high probability, increasing the height will not decrease the number of close predecessors.

Lemma B.4. *Let h be a random function such that x has t predecessors, for $t \geq \frac{u^2}{c2\pi}$. Then the probability that x has at least $\frac{u^2}{c2\pi}$ predecessors of length at most u is at least $\frac{\pi-3}{3(c-1)^2}$.*

Proof. Consider a subset of t elements of X, and consider the subset of random functions such that these t elements are the predecessors of x. If we choose a random subset of m of these predecessors and form these elements into a tree, then regardless of the shape of this tree, there are exactly the same number of ways to attach the remaining $t - m$ elements to form a larger tree. To see this, once we select the m *labelled* elements and arrange them into a tree, we can view them as m isolated points to which we attach disjoint trees formed from the remaining $t-m$ points. Each unique tree structure for the m points produces a valid and unique tree for all t points, and any such tree with the m selected points forming a subtree can be constructed in this way.

Thus, among trees where these m elements form a connected subtree rooted at x, the number of trees where these particular m elements form any particular tree shape is the same as any other tree shape.

Take any function h with a tree of t predecessors of x. Choose any m-element subset of these elements that form a connected tree rooted at x, with m such that $c\sqrt{2\pi m} = u$. By [31], the probability of this tree having height greater than u is at most

$$\frac{(\pi - 3)}{3(c-1)^2}. \tag{30}$$

If these elements have a height less than this, then they are all at most u-predecessors of x. Since this reasoning would work for any set of t predecessors, this gives the result. □

Lemma B.4 is somewhat conservative, since the number of close predecessors may grow as the tree size increases. This remains an interesting open question.

This gives us the result we need for the fraction of marked vertices in a function that we know gives many predecessors to the golden collision.

Theorem B.2. *Suppose h is a random function such that x has at least t predecessors and y has at least s predecessors. Then with probability at least $\frac{2(\pi-3)}{2(c-1)^2}\left(1+\mathcal{O}(\frac{s}{N})\right)$ the fraction of marked vertices, when iterating h at least u times, is*

$$\Omega\left(\frac{R^2\min\{u^2,t\}\min\{u^2,s\}}{N^2}\right) \tag{31}$$

Proof. Suppose h is such that x has exactly $k_x \geq t$ predecessors. If $k_x \leq u^2$, then by Lemma B.3, with the probability given, all k_x predecessors will be at a distance of at most u. Thus, every predecessor is sufficient and we have a $k_x/N \geq t/N$ probability of choosing such an element.

If $k_x > u^2$, i.e., $k_x = \frac{u^2}{c2\pi}$ for some c, then by Lemma B.4, with at least the same probability, we have at least $\frac{u^2}{c2\pi}$ predecessors of distance at most u, and hence we have a probability of $\frac{u^2}{c2\pi}$ of choosing such an element.

This also holds for y. The result follows by the same logic as Theorem B.1. Since the number of predecessors was arbitrary in this reasoning, this holds for any random function where x and y have at least t and s predecessors. □

Our only remaining issue is ensuring that the predecessors leading to x and y are detected. If we retain the last distinguished point, we will only detect them if we reach a distinguished point after the golden collision. This is a property of the function h; if the next distinguished point is too far, then *all* predecessors of x and y will fail to detect the collision.

Thus, suppose that we iterate h for $u_1 + u_2$ times. We choose u_1 to optimize the bounds in the previous theorems, assuming that after roughly u_1 steps we reach the golden collision. We choose u_2 to reach a distinguished point.

Each iteration after the golden collision has a θ chance of being a distinguished point. Thus, the probability of missing a distinguished point is $(1-\theta)^{u_2} < e^{-\theta u_2}$, so $u_2 = \Omega(1/\theta)$ gives a constant probability that a particular function will reach a distinguished point within n_2 steps after the golden collision.

Ultimately, this leads to our main theorem:

Theorem B.3. *Let $1 \leq t$ be in $\mathcal{O}(1/\theta)$. Then with probability $\Omega(\frac{1}{t})$, the fraction of marked vertices is $\Omega(\frac{R^2\min\{u^4,t^2\}}{N^2})$.*

Proof. From Lemma B.1, the probability is $\Theta(\frac{1}{t})$ that both halves of the golden collision will have at least t predecessors. Theorem B.2 shows that a constant proportion of these functions will have at least $\Omega(\frac{R^2\min\{u^4,t^2\}}{N^2})$ marked vertices. □