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, 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 [NIS16], 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) [JAC$^+$17,ACVCD$^+$,CLN$^+$]. They can also be used in certain lattice attacks [APS15].

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 the 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 [ACVCD+]. The authors concluded that the most efficient method was van Oorschot-Wiener's distinguished point technique [vOW99]. 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. In a quantum circuit, a variety of basic *quantum gates* are applied to (logical) *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.

Then, the best quantum algorithm for golden collision search is Ambainis' algorithm [Amb07], which would find the collision in time $\widetilde{\mathcal{O}}(N^{2/3})$ if $|X| = N$, matching a query lower bound of $\mathcal{O}(N^{2/3})$ [AS04]. However, this algorithm 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 consider that a memory access to R qubit registers requires $\Theta(R)$ operations or quantum gates. This means simply going back to the baseline circuit model, as done *e.g.* in [BBG+13] in the context of distributed quantum algorithms, and previously in [JS19] for a study of known quantum algorithms for golden collision search. This makes all memory accesses very costly. 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 [JS19]. 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 take into account locality constraints. In particular, we consider the model of a two-dimensional grid with local interactions only. We also give costs for non-local models, mainly for comparison.

Contributions. Our first objective is to obtain the best *gate count* for this problem. We do that in Section 3. We first rewrite van Oorschot-Wiener collision search in the random walk framework, and obtain a quantum analogue in the MNRS *quantum walk* framework, that uses iterates of the function h. If h is a single gate evaluated in time 1, its gate and time complexity is of $\mathcal{O}(N^{6/7})$. Next, we give another algorithm that does not iterate h but searches for distinguished points with Grover's search. Surprisingly, these two different methods achieve the exact same complexity; we explain why this is the case. This requires a more thorough analysis of the random functions that govern the run-time of van Oorschot-Wiener collision search, which may be of independent interest.

In Section 4, we give a parallel version of our prefix-based walk, and a parallel multi-Grover search algorithm that improves over [BBG$^+$13]. This gives the G-cost and DW-cost of our algorithms under depth constraints, improving on the counts of [JS19].

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 [CNS17], 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 [JS19], but they still meet the NIST security levels claimed in [JAC$^+$17].

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 standard *quantum circuit model* (see *e.g.* [NC02]). This abstract model of computation underlies most of the physical architectures currently under study. In order to give meaningful cost estimates of quantum circuits, we use the memory peripheral model of [JS19]. This means 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 use then two cost metrics depending on physical assumptions:

– 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, and we show in Section 3.6 that our best gate counts can be obtained with a two-dimensional grid mesh with no increase in gate cost or depth. In Section 4, this layout plays a role, so we consider the two following alternatives: either the grid, or no locality issues. It is shown in [BBG+13] that the general quantum circuit model can be emulated by any network allowing sufficient connectivity, such as a hypercube, with only polynomial overhead, but a two-dimensional grid does not have sufficient connectivity for this.

Quantum Memory Models. Many quantum algorithms require the "qRAM" model, in which the access *in superposition* to the elements in memory is a cheap operation. But qRAM is not a feature of the quantum circuit model, and it must come with a specific physical architecture whose realizability is unclear at the moment[3]. This model can be restricted to *quantum-accessible classical memory* (QRACM, see [Kup13, Section 2]), while the best time complexities for golden collision search [Amb07,Tan09] require QRAQM, that is, the states in the memory accessed are also quantum.

Both QRAQM and QRACM can be constructed in the quantum circuit model with Clifford+T gates with no special hardware assumptions. 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 $\mathrm{poly}(R)$ 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). Below we recall that an algorithm for golden collision search can easily be adapted to *single collision search* (Problem 2.2), *element distinctness* (Problem 2.3) and *claw-finding* (Problem 2.4).

Problem 2.1 (Golden collision finding). Let $h : X \to X$ be a 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*. Find a golden collision.

In many instances there will be a unique golden collision. We assume h is a pseudo-random function. If not, we can pick a random function $f : X \to X$ and compose it with h. If h is a permutation, the composition will be a random function, but otherwise it will not. However, we assume that h is sufficiently similar to a permutation that $f \circ h$ will be pseudo-random. Practically, this means we assume that h does not have a serious restriction on its outputs.

Problem 2.2 (Single collision). Given access to a random function $H : \{0,1\}^n \to \{0,1\}^m$ where $m \geq 2n$, find a collision of H if it exists.

[3] See [GLM08,AGJO+15] for the "bucket-brigade" architecture, which still requires $\Theta(R)$ gates for a memory access to R bits of memory.

We can choose a random function $f : \{0,1\}^m \to \{0,1\}^n$, and then $f \circ H : \{0,1\}^n \to \{0,1\}^n$ acts like the function h in the golden collision finding problem. Our choice of f is likely to produce many extra collisions, so we check each collision under H to see if it collides in $\{0,1\}^m$; this acts as the check function g.

Problem 2.3 (Element distinctness). Given $h : \{0,1\}^n \to \{0,1\}^n$, determine if h is a permutation or not.

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

Problem 2.4 (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 denote the cost of evaluating h by H and the cost of g by 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. We will use a capital R to denote memory size. Typically this will refer to words of memory, such as n-bit strings that represent the inputs or outputs of h.

Previous algorithms. We assume that the functions 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 [vOW99]. 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 $R = N$.

Quantum algorithms for the problem started with Buhrman *et al.* [BDH+05], who give an algorithm in $\widetilde{\mathcal{O}}(N^{3/4})$ quantum time and $\mathcal{O}(N^{1/2})$ memory for claw-finding and element distinctness. Ambainis [Amb07] 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 [AS04]. Tani provided a claw-finding version [Tan09].

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's algorithm also costs $\widetilde{\mathcal{O}}(N)$ gates. A careful analysis shows that, if evaluating the function h costs H gates, Tani's algorithm provides a $\mathcal{O}(\sqrt{\mathsf{H}})$ advantage over Grover's algorithm [JS19].

Golden collision finding can be seen as a unique 2-XOR problem. For $k = 2$, the optimization program given in [NS20] recovers the algorithm of [BDH$^+$05].

Another approach based on a distributed computing model achieves a very good time-memory tradeoff of $TM = \widetilde{\mathcal{O}}(N)$ [BBG$^+$13]. 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. Further, there are locality issues; achieving this time-memory tradeoff requires a network that can sort itself in poly-logarithmic time.

The distributed algorithm for multi-target preimage search given in [BB18] can also be reframed for golden collision search, in which case it becomes a variant of [BBG$^+$13] based on iterating a random function and computing "chain-ends" (instead of using a parallel RAM emulation unitary). But it is an inherently parallel algorithm and it does not reach a smaller gate cost than $\widetilde{\mathcal{O}}(N)$.

Random Collision Search. When $h : \{0,1\}^n \to \{0,1\}^n$ is a random function, a collision can be found in classical time $\mathcal{O}(2^{n/2})$. Brassard *et al.* [BHT98] give a quantum algorithm with time $\widetilde{\mathcal{O}}(N^{1/3})$, using a QRACM of size $\mathcal{O}(N^{1/3})$ (classical memory with quantum random access). This requirement has later been discussed by multiple authors [GR04,Ber09]. In the quantum circuit model, the lowest gate-count to date is obtained with the algorithm of [CNS17]. The algorithm has a gate complexity of $\mathcal{O}(N^{2/5})$ with $\mathcal{O}(N^{1/5})$ *classical memory without quantum random access*, and makes a total of $\mathcal{O}(N^{1/5})$ accesses to the memory.

3 Golden Collision Finding with Random Walks

In this section we briefly define random walk search, both classical and quantum. We then reframe van Oorschot-Wiener parallel collision search [vOW99] as a random walk, and provide a quantum analogue that is one of our gate-optimal algorithms. The other gate-optimal algorithm is also a random walk, and we compare the two.

3.1 Random Walk Search

Let $G = (V, E)$ be an undirected, connected, regular graph. We suppose there is some subset of vertices M, which we call "marked" vertices, 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_i.
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, Nayak, Roland, and Santha (MNRS) present an algorithm and unified framework to solve such tasks [MNRS11]. 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.

Classical Random Walk. We will describe a classical random walk. Not only will this provide intuition for the quantum random walk, but we will later show that vOW parallel collision search is equivalent to such a 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.

Taking $\mathcal{O}(\frac{1}{\delta})$ random steps will reach the stationary distribution of the graph. Because we assumed a regular graph, this is the uniform distribution. Hence, sampling from this distribution has a $\frac{1}{\epsilon}$ chance of returning a marked vertex. 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 almost entirely analogous to the classical case, in the same way that Grover's search algorithm [Gro96,BHMT02] is analogous to a brute force search. The cost of the quantum random walk is

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

If we use the Tolerant Recursive Amplitude Amplification technique from MNRS [MNRS11], possibly using a qubit as control, then we can find the marked vertex in $\mathcal{O}(1/\sqrt{\epsilon})$ iterations when ϵ is only a lower bound on the fraction of marked vertices.

Johnson Graphs. We consider random walks on Johnson graphs. A Johnson graph on a set X of size R is a graph whose vertices are all subsets of X of size R, and two vertices are adjacent if they differ in exactly one element. The spectral gap of a Johnson graph is $\delta = \Omega(\frac{1}{R})$. A random step is equivalent to selecting a random element from the current vertex (which is a set) and deleting it, then selecting a new random element and inserting it into the set.

To efficiently represent these 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 [JS19].

Errors in Random Walks. We will encounter two cases for the update procedure U. In Section 3.4, we will 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 instead 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. Section A studies this in more detail (see Theorem A.1).

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

$$\mathcal{O}\left(\mathsf{S} + \frac{1}{\sqrt{\epsilon}}\left(\frac{1}{\sqrt{\delta}}\mathsf{U} + \mathsf{C}\right)\right) \text{ to } \mathcal{O}\left(\mathsf{S} + \frac{1}{\sqrt{\epsilon'}}\left(\frac{1}{\sqrt{\delta}}\mathsf{U} + \mathsf{C}\right)\right) \tag{3}$$

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

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

3.2 Ambainis' Algorithm

Ambainis' element distinctness algorithm [Amb07] performs a random walk on a Johnson graph of size R on the domain of the function h. This is a query-optimal algorithm for Problem 2.1.

Classical Walk. We store elements as tuples $(x, h(x))$. The list is sorted according to $h(x)$ to make it easy to check, when inserting or deleting an element, if there are any collisions.

We denote a vertex as *marked* if it contains both the elements x_g and y_g which form the golden collision. Each vertex has a single flag indicating if is marked. The fraction of marked vertices is the number of sets containing the 2 fixed elements x_g and y_g, which will be

$$\epsilon = \frac{R(R-1)}{N^2}. \tag{4}$$

It will cost $\mathcal{O}((\mathsf{H} + \log R)R)$ to initialize the list, in sorted order.

To take a single random step, we incur a cost of $\mathsf{H} + \log R$ to compute a new element and insert it into the list; however, we will also check for collisions with existing elements in the list and update the flag. 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,

$$\mathsf{U} = \mathsf{H} + \log R + \frac{R-1}{N}\mathsf{G}. \tag{5}$$

Because we update the flag in the update step, the check step only needs to check the flag, at cost $\mathcal{O}(1)$.

As a classical random walk, this gives a total cost of

$$\mathcal{O}\left(R(\mathsf{H} + \log R) + \frac{N^2}{R(R-1)}\left(R\left(\mathsf{H} + \log R + \frac{R-1}{N}\mathsf{G}\right) + 1\right)\right). \tag{6}$$

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 optimal, which produces a cost of roughly $\mathcal{O}(N\mathsf{H})$.

Quantum Variant. *Assuming cheap QRAQM* (or a unit cost for the "qRAM gate", as formulated in [Amb07]) the costs and parameters from before are essentially the same for the quantum case. This gives the following complexity:

$$\mathcal{O}\left(R(\mathsf{H} + \log R) + \frac{N}{\sqrt{R(R-1)}}\left(\sqrt{R}\left(\mathsf{H} + \log R + \frac{R-1}{N}\mathsf{G}\right) + 1\right)\right). \tag{7}$$

We optimize this by taking $R = N^{2/3}$, for a total cost of $\widetilde{\mathcal{O}}(N^{2/3})$. There are subtle issues ensuring that each subroutine is reversible and constant-time, but we ignore those for now.

Costing Memory. With only one- and two-qubit gates, a single memory access to R elements costs $\Theta(R)$ gates. 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. This changes the update cost to

$$\mathsf{U} = \mathsf{H} + R + \frac{R(R-1)}{N}\mathsf{G}, \tag{8}$$

leading to a total cost of

$$\mathcal{O}\left(R(\mathsf{H} + R) + \frac{N}{\sqrt{R(R-1)}}\left(\sqrt{R}\left(\mathsf{H} + R + \frac{R(R-1)}{N}\mathsf{G}\right) + 1\right)\right). \tag{9}$$

Here, the optimal occurs when $R = \mathsf{H}$, for a total cost roughly $\mathcal{O}(N\sqrt{R}) = \mathcal{O}(N\sqrt{\mathsf{H}})$.

Previous work [JS19] noticed that Grover's algorithm has gate cost of $\mathcal{O}(N\mathsf{H})$, so Tani's algorithm [Tan09] and Ambainis' algorithm [Amb07] 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 collision finding algorithm, but 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 [vOW99] is to "lift" the function h via *distinguished points*. We select a random subset X_D and denote such points as "distinguished". In practice we choose bitstrings with a fixed prefix. We then lift the random function $h : \{0,1\}^n \to \{0,1\}^n$ to a random function $h_D : \{0,1\}^n \to X_D$. Then 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. If $|X_D| = \theta N$, 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, 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 Appendix). For now, we will simply say that $u \approx 1/\theta$.

With h_D constructed, we build the same random walk as Ambainis' algorithm. We use the same Johnson graph $J(X, R)$, but we store elements as $(x, h_D(x), u_x)$, where u_x is such that $h_D(x) = h^{u_x}(x)$. We then mark vertices if and only if they contain x and y that are on a "trail" leading to the golden collision. This means that there are integers i and j with $0 \leq i \leq u_x$ and $0 \leq j \leq u_y$ such that $h^i(x) = x_g$ and $h^j(y) = y_g$.

To perform a random walk, the update step costs $u\mathsf{H} = \mathcal{O}(\mathsf{H}/\theta)$ to compute $h_D(x)$ for a random insertion of x, and 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. We calculate the average number of collisions in the list as follows: 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 total probability of a collision is u^2/N for each point in the list, and so the average number of collisions is $(R-1)u^2/N = \mathcal{O}(R/N\theta^2)$.

Thus, the update cost becomes

$$\mathsf{U} = \mathcal{O}\left(\frac{\mathsf{H}}{\theta} + \log R + \frac{R}{N\theta^2}\frac{\mathsf{H}}{\theta} + \frac{R}{N\theta^2}\mathsf{G}\right). \tag{10}$$

From here on we assume that $\mathsf{G} \ll u\mathsf{H}$, so we ignore the last term.

Section B gives a detailed analysis of the number of marked elements. Roughly speaking, every random function will produce some number of *predecessors* to each half of the golden collision. These are points z such that $h^k(z) = x_g$ for some k. In order for a vertex to be marked, we must select at least one predecessor for each half of the golden collision when we select the R random starting points for elements of the vertex. Thus, 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.

We show (Lemma B.2) 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 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).

We put this together to get the classical cost. Assume $\log R \ll \frac{\mathsf{H}}{\epsilon}$, and that H/θ dominates G, then the cost of a single walk is:

$$\mathcal{O}\left(R\left(\mathsf{H}n + \log R\right) + \frac{N^2}{R^2t^2}\left(R\left(\frac{\mathsf{H}}{\theta} + \log R + \frac{(R-1)n^2}{N}\frac{\mathsf{H}}{\theta}\right) + 1\right)\right) \tag{11}$$

$$= \mathcal{O}\left(\frac{R\mathsf{H}}{\theta} + \frac{N^2}{Rt^2\theta}\mathsf{H} + \frac{N}{t^2\theta^3}\mathsf{H}\right). \tag{12}$$

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

$$\mathcal{O}\left(\frac{tR\mathsf{H}}{\theta} + \frac{N^2}{Rt\theta}\mathsf{H} + \frac{N}{t\theta^3}\mathsf{H}\right). \tag{13}$$

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 $\theta = \sqrt{R/N}$. Substituting, we get

$$\mathcal{O}\left(t\sqrt{NR}\mathsf{H} + \frac{N^{5/2}}{R^{3/2}t}\mathsf{H}\right). \tag{14}$$

This is balanced 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 Variant

As with Ambainis' algorithm, the costs will remain approximately the same in the quantum case, except for three main differences: The cost to access memory is now $\mathcal{O}(R)$, the $1/\epsilon$ and $1/\delta$ terms get square root speed-ups, and we perform a Grover search among random functions. The random walk will act as the oracle for the Grover search, which will thus need to repeat the oracle $\mathcal{O}(\sqrt{t})$ 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, they will form a tree, which has height $\sqrt{2\pi t}$ on average. Thus, many predecessors are useless because a trail that starts too high will not reach the golden collision. However, 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 be $\Omega(\frac{R^2}{N^2\theta^4})$.

This gives a total cost of

$$\mathcal{O}\left(t^{\frac{1}{2}}\left(\frac{R\mathsf{H}}{\theta} + \frac{N\theta^2}{R}\left(R^{\frac{1}{2}}\left(\frac{\mathsf{H}}{\theta} + R + \frac{(R-1)\mathsf{H}}{N\theta^3}\right) + 1\right)\right)\right) \tag{15}$$

$$= \mathcal{O}\left(\frac{t^{\frac{1}{2}}R\mathsf{H}}{\theta} + \frac{Nt^{\frac{1}{2}}\theta}{R^{\frac{1}{2}}}\mathsf{H} + N(Rt)^{\frac{1}{2}}\theta^2 + \frac{(Rt)^{\frac{1}{2}}}{\theta}\mathsf{H}\right) \tag{16}$$

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:

$$\mathcal{O}\left(N^{6/7}\mathsf{H}^{5/7}\right) \tag{17}$$

This result shows that even the most costly model of quantum memory access brings an advantage in the G-cost.

3.5 Prefix-based Walk

We now present an alternative quantum walk that uses Grover search to directly find distinguished points in X_D. We assume that the golden collision happens among distinguished points. To ensure this happens, we can compose with a random function or choose different prefixes to form new arbitrary definitions of X_D. Each new definition of X_D has a probability of θ of containing the golden collision, thus after $\frac{1}{\theta}$ trials (or $\frac{1}{\sqrt{\theta}}$ Grover iterates) we expect the golden collision to be among the distinguished points.

Then, given a choice of X_D, we perform a quantum walk similar to Ambainis' algorithm, but only among X_D.

A vertex in the walk contains R elements $(x, h(x))$ where $h(x) \in X_D$, which are sorted by output value $h(x)$. A vertex contains a counter indicating the number of golden collisions it has found. We use the Johnson vertex structure [JS19] again for fast insertion, search, and collision detection. The analysis is very similar to Ambainis' algorithm, except that the setup and update procedures, instead of creating a uniform superposition over X, create a superposition over X_D using a Grover search in time $\mathsf{H}/\sqrt{\theta}$.

A technical difficulty is that the precise size of X_D cannot be estimated at runtime with arbitrary precision. Instead, we assume a fixed size θN. Due to this, there will be a negligible probability of error, depending on the difference between $|X_D|$ and θN for the actual good choice of distinguished points.

Gate Complexity. The setup costs $R\frac{\mathsf{H}}{\sqrt{\theta}} + R\log R$, the update costs $\frac{\mathsf{H}}{\sqrt{\theta}} + R$. The checking is trivial. If we assume that X_D is a good choice, the probability that a vertex contains the golden collision is $R^2/(\theta^2 N^2)$. If this is not, the golden collision will not be found. So after $N\theta/R$ iterations of the quantum walk, we check whether the current vertex is marked or not. The total cost for a single walk is:

$$\mathcal{O}\left(R\frac{\mathsf{H}}{\sqrt{\theta}} + R\log R + \frac{N\theta}{R}\left(\sqrt{R}\left(\frac{\mathsf{H}}{\sqrt{\theta}} + R\right) + 1\right)\right) . \tag{18}$$

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. $\theta \geq (\mathsf{H}/N)^{2/3}$. The total gate cost, with the Grover search, is:

$$\mathcal{O}\Bigg(\underbrace{\frac{1}{\sqrt{\theta}}}_{\text{Search}}\Bigg(\underbrace{\frac{\mathsf{H}^2}{\theta}}_{\text{Setup}} + \underbrace{N\theta^{3/4}\sqrt{\mathsf{H}}}_{\text{Walk}}\Bigg)\Bigg) = \mathcal{O}\left(\mathsf{H}^2\theta^{-3/2} + N\theta^{1/4}\sqrt{\mathsf{H}}\right) . \tag{19}$$

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 $\mathcal{O}(N\theta^{1/4}\sqrt{\mathsf{H}}) = \mathcal{O}(N^{6/7}\mathsf{H}^{5/7})$ and corresponding memory $R = N^{2/7}\mathsf{H}^{4/7}$. So minimizing the gate cost gives exactly the same result as in Section 3.3.

3.6 Comparison

Both the prefix-based walk and the iteration-based walk use distinguished points to improve the search. They differ in how they find distinguished points, whether by a direct search for the prefix or by iterating. Classically, the two approaches have the same asymptotic cost to find a single distinguished point, but 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.

Furthermore, both approaches must repeat the underlying random walk. The iteration-based search must span many functions to ensure that the desired collision has a large set of predecessors; the prefix-based search must redefine the set of distinguished points to ensure that it will contain the golden collision.

In concrete terms, for the correct definition of distinguished points, a prefix-based search walks on a graph with $\Omega(\frac{R^2}{N^2\theta^2})$ marked vertices, while an iteration-based search walks on a graph with $\Omega(\frac{R^2}{N^2\theta^4})$ marked vertices. The extra powers of θ reflect the higher chance of collision on trails. However, there are only $1/\theta$ possible prefixes to search through, while an iteration-based search must search $\mathcal{O}(1/\theta^2)$ functions to find one that gives enough predecessors.

Classically, this gives advantage to iteration-based methods, with an overall factor of $\mathcal{O}(\theta^2)$, rather than $\mathcal{O}(\theta)$ for prefix-based search. The quantum iteration-based method retains an advantage of $\mathcal{O}(\sqrt{\theta})$ in the number of walk *steps*, but each step costs an extra factor of $\mathcal{O}(1/\sqrt{\theta})$. This advantage and disadvantage cancel out, giving our result that both methods asymptotically cost $\mathcal{O}(N^{6/7}\mathsf{H}^{5/7})$ gates.

Time costs and locality. In our algorithms, we can assume that memory access has an $\mathcal{O}(R^{1/2})$ time cost, reflecting either latency or locality in a two-dimensional layout. Substituting this into Equation 16 or 19 does not change the time, as we already pay a time R in the update procedure, in order to find a new element to insert.

For prefix-based walks, Grover search is easily local, and the set-up step can be done by initializing the elements in time $\mathcal{O}(R\mathsf{H}/\sqrt{\theta})$, then sorting them in time $\mathcal{O}(R^{3/2})$. Similar logic applies to the set-up of the iteration-based walk. Section A describes how the iterations can also be local. Hence, both algorithms achieve the same complexity with local connectivity.

4 Parallelization

In Section 3, we optimized only the *gate cost* and found algorithms which benefited from leaving most of the qubits idle for most of the time. In other contexts, we may try to apply more simultaneous gates at each step to reduce the total depth, which may or may not increase gate complexity. For example, the memory access circuit may use $\mathcal{O}(R)$ gates sequentially, or use a tree of depth $\mathcal{O}(\log R)$, and the gate cost is the same. In contrast, the depth of a Grover search can be reduced by a factor $\sqrt{P}$, but this increases its gate cost by the same factor.

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 [BBG+13], 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 $\mathsf{D} \leq N^{1/2}$ with gate cost $\mathcal{O}(N^{3/2}/\mathsf{D})$.

4.1 Prefix-based Walk

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 introduce a parameter P. For example, a Grover search can be distributed on P processors with a reduction of a factor $\sqrt{P}$ in depth and an increase of a factor $\sqrt{P}$ in gate cost. However, gates that can applied simultaneously (*e.g.* in sorting networks) do not add more depth to the circuit.

We consider the algorithm of Section 3.5. We repeat for $\frac{1}{\sqrt{\theta}}$ iterations a walk that searches for a golden collision with a given definition of distinguished points. The setup can be perfectly parallelized. We do not parallelize the iterations of the walk, and instead use our computing power to accelerate each update step. The depth to find an element with a good prefix can be reduced to $\frac{\mathsf{H}_D}{\sqrt{\theta}\sqrt{P}}$ by parallelizing the Grover search, as long as we have $P \leq R$ (otherwise we are using too much memory). We also need $P \leq \frac{1}{\theta}$. This increases the total gate cost to

$$\mathcal{O}\left(\frac{R\mathsf{H}_G}{\theta} + \frac{\mathsf{S}_G}{\sqrt{\theta}} + \frac{N\sqrt{\theta}}{R}\left(\sqrt{R}\left(\frac{\mathsf{H}_G\sqrt{P}}{\sqrt{\theta}} + R\right) + 1\right) \right) \tag{20}$$

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 $P\theta \leq 1$. By replacing R in this equation we find $\theta = N^{-4/7}\mathsf{H}_G^{6/7}P^{1/7}$. This gives $R = \mathsf{H}_G^{4/7}N^{2/7}P^{3/7}$, and the condition of $P\theta \leq 1$ becomes $P^8\mathsf{H}_G^6 \leq N^4$. The total gate cost becomes

$$\mathcal{O}(N^{6/7}\mathsf{H}_G^{5/7}P^{2/7}). \tag{21}$$

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

$$\mathcal{O}\left(\frac{\mathsf{H}_D}{\sqrt{\theta}} + \mathsf{S}_D + \frac{N\theta}{R}\sqrt{R}\left(\frac{\mathsf{H}_D}{\sqrt{\theta}\sqrt{P}} + \mathsf{S}_D\right)\right) . \tag{22}$$

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. In the non-local setting, we can parallelize up to $P = \mathcal{O}(N^{1/2})$ and with a two-dimensional mesh we can reach $P = \mathcal{O}(N^{4/11})$.

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.

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 Iteration-based walk

Parallelizing the vOW search works well because different processors can independently iterate the hash function. In the quantum analogue, after we insert a new element into the list, we must uncompute another element; this uncomputation seems to need to be serial. Thus, the classical parallelization does not apply.

However, if we simply task P processors to iterate the hash function for $\mathcal{O}(1/P\theta)$ iterations, we expect one of them to produce a distinguished point. We thus reduce the time to find a distinguished point, but the distinguished points

Table 1: Asymptotic costs and parameters for prefix-based random walks. For readability, terms of H and asymptotic notation are omitted. The results depend whether locality is assumed or not. The line “Any” describes a tradeoff for any D, until $\mathsf{D} = N^{1/2}$ in the non-local case and $\mathsf{D} = N^{8/11}$ in the local case (2-dimensional grid with nearest-neighbor connectivity). “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		Depth	DW-cost
				Inner	Outer		
Any	No (Sec. 3)	$N^{\frac{6}{7}}$	$N^{\frac{2}{7}}$	1	1	$N^{\frac{6}{7}}$	$N^{\frac{8}{7}}$
	D	$N^{\frac{6}{5}}\mathsf{D}^{-\frac{2}{5}}$	$N^{\frac{4}{5}}\mathsf{D}^{-\frac{3}{5}}$	$N^{\frac{6}{5}}\mathsf{D}^{-\frac{7}{5}}$	1	D	$N^{\frac{4}{5}}\mathsf{D}^{\frac{2}{5}}$
Non-local	$\mathsf{D} = N^{\frac{1}{2}}$	N	$N^{\frac{1}{2}}$	$N^{\frac{1}{2}}$	1	$N^{\frac{1}{2}}$	N
	$\mathsf{D} \leq N^{\frac{1}{2}}$	$N^{\frac{3}{2}}\mathsf{D}^{-1}$	$N^{\frac{1}{2}}$	$N^{\frac{1}{2}}$	$N\mathsf{D}^{-2}$	D	$N^{\frac{3}{2}}\mathsf{D}^{-1}$
2-dim. neighbors	$\mathsf{D} = N^{\frac{8}{11}}$	$N^{\frac{11}{12}}$	$N^{\frac{4}{11}}$	$N^{\frac{2}{11}}$	1	$N^{\frac{8}{11}}$	$N^{\frac{12}{11}}$
	$\mathsf{D} \leq N^{\frac{8}{11}}$	$N^{\frac{18}{11}}\mathsf{D}^{-1}$	$N^{\frac{4}{11}}$	$N^{\frac{2}{11}}$	$N^{\frac{16}{11}}\mathsf{D}^{-2}$	D	$N^{\frac{20}{11}}\mathsf{D}^{-1}$

we find have very short trails. Short trails are less likely to collide. We analyzed this method and found that it is strictly worse than parallelizing the prefix-based method.

A different method would be to stagger the iteration process, so each processor is $1/P\theta$ steps ahead of the next one. After $1/P\theta$ steps, one of the processors has finished $1/\theta$ total iterations and likely has a distinguished point ready to insert into the list. The problem now is that once we have inserted an element, we must uncompute the insertion operation for an element we will delete. Naively, these operations do not commute, so we must perform the computation and uncomputation sequentially, preventing us from precomputing any of the function iterations. If these operations commute, it would allow near-perfect parallelization of the iteration-based walk.

4.3 Multi-grover Search

While random walks provide the lowest gate cost when depth is unlimited, practical constraints make parallelization more important. Our next algorithm is a prefix-based adaptation from [BBG+13].

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 will wrap the entire algorithm in a Grover search for the correct prefix, which will require $\mathcal{O}(1/\sqrt{\theta})$ iterations.

Given P processors, we use each one to separately search for x such that $h(x)$ is a distinguished point. This has cost $\mathcal{O}(\mathsf{H}/\sqrt{\theta})$ per processor, so the total cost is $\mathcal{O}(\mathsf{H}P/\sqrt{\theta})$.

We then treat the processors as a sorting network, and sort based on the value of $h(x)$. We check each pair of colliding elements for the golden collision, and propagate the result with a tree structure (such as an H-tree). Then we unsort the list so we can uncompute the list.

If we have chosen the prefix correctly, then the golden collision will be two of $N\theta$ points. The Grover search will produce a random list of P points out of those $N\theta$, so the probability of containing the golden collision is at least

$$\frac{\binom{P}{2}(N\theta)^{P-2}}{(N\theta)^P} = \Omega\left(\frac{P^2}{N^2\theta^2}\right). \tag{23}$$

We then perform a Grover search over these lists. This leads to a total cost of

$$\mathcal{O}\left(\frac{1}{\theta^{1/2}}\frac{N\theta}{P}\left(\frac{\mathsf{H}P}{\theta^{1/2}}+S_G\right)\right) = \mathcal{O}\left(N\mathsf{H}+\frac{N\theta^{1/2}S_G}{P}\right) \tag{24}$$

The sorting cost S_G is the interesting factor here. If S_G/P is small, then the $\mathcal{O}(N\mathsf{H})$ term will be greatest and lead to a near-perfect parallelization. This is the original result of [BBG+13]. 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 $\theta = \mathsf{H}^2/P$.

For depth in this case, the depth to construct each list is $\mathcal{O}(\mathsf{H}/\theta^{1/2})$ and we denote the depth to sort as S_D, so the total depth is

$$\mathcal{O}\left(\frac{N\theta^{1/2}}{P}\left(\frac{\mathsf{H}}{\theta^{1/2}}+S_D\right)\right) = \mathcal{O}\left(\frac{N\mathsf{H}}{P}+\frac{N\theta^{1/2}S_D}{P}\right). \tag{25}$$

In the two-dimensional mesh, $S_D = \mathcal{O}(P^{1/2})$, so with the same value of $\theta = \mathsf{H}^2/P$ we find a total depth of $\mathcal{O}(N\mathsf{H}/P)$. Thus, this algorithm parallelizes perfectly, even accounting for locality and communication costs.

The maximum parallelization we can achieve by this method is $P = \mathcal{O}(N^{1/2})$. At this point, there are $\mathcal{O}(N^{1/2})$ elements where $h(x)$ has the fixed prefix, and our list has $\mathcal{O}(N^{1/2})$ elements, so there is a constant probability of containing the golden collision. Beyond this point, the same parameterization would increase the cost quadratically in P. For lower depths, we simply divide the search space. Table 2 summarizes these results.

If we have some architecture where $S_D = o(P^{1/2})$, then we can choose $\theta = P/N$ and the asymptotic depth is $\mathcal{O}(N\mathsf{H}/P)$ even for very large P.

5 Quantum (Parallel) Collision Search

In this section, we study the algorithm of [CNS17] 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, we take a random function $h : \{0,1\}^n \to \{0,1\}^n$ that has expectedly many collisions, and the goal is to output one. We improve the parallelization given in [CNS17] in order to achieve the best gate counts under a depth restriction. This will help us comparing the complexity of our golden collision search algorithms to the desired security levels.

Table 2: Prefix-based Multi-Grover on a local architecture. "P processors" gives the cost with $P \leq N^{1/2}$ processors. "Depth limit $\mathsf{D} \geq N^{\frac{1}{2}}$ ' gives the cost with the minimum number of processors to fit a depth limit D. "Fastest single" gives the minimum depth achievable with the Multi-Grover parallelization. Below this depth, we divide the search space to parallelize, giving "Depth limit $\mathsf{D} \leq N^{\frac{1}{2}}$".

Parameters	G-cost	Total hardware	Depth	DW-cost
P processors	N	P	NP^{-1}	N
Depth limit $\mathsf{D} \geq N^{\frac{1}{2}}$	N	$N\mathsf{D}^{-1}$	D	N
Fastest single	N	$N^{\frac{1}{2}}$	$N^{\frac{1}{2}}$	N
Depth limit $\mathsf{D} \leq N^{\frac{1}{2}}$	$N^{\frac{3}{2}}\mathsf{D}^{-1}$	$N^{\frac{3}{2}}\mathsf{D}^{-2}$	D	$N^{\frac{3}{2}}\mathsf{D}^{-1}$

Algorithm. The algorithm of [CNS17] also relies on the definition of *distinguished points* (d.p.) via arbitrary prefixes. It runs in two phases: first, M distinguished points are found, using Grover's algorithm (where the proportion of distinguished points is θ). These elements are stored in a *classical* memory with *sequential access*. Second, we look for a collision on the distinguished points, using quantum search. An iteration of the search must build the superposition of distinguished points and then test if they belong to the stored memory. This is done with a sequential circuit. Hence, the memory is accessed only once per iteration of the search, and it does not need to be random-access. The gate complexity is:

$$M\frac{\mathsf{H}_G}{\sqrt{\theta}} + \sqrt{\frac{N\theta}{M}}\left(\frac{\mathsf{H}_G}{\sqrt{\theta}} + M\right)$$

and the gate count is optimal when $\frac{\mathsf{H}_G}{\sqrt{\theta}} = M$ and $M^2 = \sqrt{\frac{N\theta}{M}}M$ *i.e.* $\theta = M^3/N$ and $M = \mathsf{H}_G^{2/5}N^{1/5}$. Then we have a gate count of $\mathsf{H}_G^{4/5}N^{2/5}$ [CNS17].

Parallelized Algorithm. The authors of [CNS17] considered a situation in which the first phase is distributed on many quantum processors, the distinguished points are then stored in a single classical memory, and the second phase is a distributed Grover search. We can do better with the algorithm of [BBG$^+$13], similar to our Multi-Grover golden collision search. Each processor has a local classical memory of exponential size, where it stores its distinguished points. In the second step, we distribute the search on the P processors, and when testing a new value of h for a collision in the stored data, we use the quantum parallel RAM emulation unitary of [BBG$^+$13, 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 becomes:

$$\mathcal{O}\left(M\frac{\mathsf{H}_G}{\sqrt{\theta}} + P\sqrt{\frac{N\theta}{MP}}\left(\frac{\mathsf{H}_G}{\sqrt{\theta}} + \frac{M}{P^2}S_G\right)\right)$$

and the total depth is:

$$\mathcal{O}\left(\frac{M}{P}\frac{\mathsf{H}_D}{\sqrt{\theta}} + \sqrt{\frac{N\theta}{MP}}\left(\frac{\mathsf{H}_D}{\sqrt{\theta}} + \frac{M}{P}S_D\right)\right) \quad .$$

If $S_G = P\log P$, then by optimizing the gate count we get $\frac{\mathsf{H}_G}{\sqrt{\theta}} = \frac{M}{P}$ and $M^2/P = \sqrt{N\theta M/P}$ i.e. $\theta = M^3/(NP)$ i.e. $M = \mathsf{H}_G^{2/5}N^{1/5}P^{3/5}$, then we have a gate count of $\mathsf{H}_G^{4/5}N^{2/5}P^{1/5}$. In general, we set $\frac{\mathsf{H}_G}{\sqrt{\theta}} = \frac{M}{P^2}S_G$ and $\theta = M^3/(NP)$ still holds, i.e. $M = \mathsf{H}_G^{2/5}N^{1/5}PS_G^{-2/5}$. Then we have a gate count of $\frac{M^2}{P^2}S_G = \mathsf{H}_G^{4/5}N^{2/5}S_G^{1/5}$, which is directly related to the gate cost of sorting on the P processors.

Assuming that $S_D = S_G/P$ and writing that $\mathsf{H}_D \leq \mathsf{H}_G$, the depth of the circuit is the previous gate count divided by P, i.e. $\mathsf{H}_G^{4/5}N^{2/5}S_G^{1/5}P^{-1}$. The parallelization from [CNS17] occurs when $S_G = P^2$, *i.e.* in the worst-case scenario for communication costs. In general, this is valid as long as $P \leq M$ and $N\theta \geq MP$, which also translates to $M \geq P$; hence $S_G^{2/5} \leq \mathsf{H}_G^{2/5}N^{1/5}$, since S_G is a function of P. For example, on a two-dimensional grid the limit is $P \leq \mathsf{H}_G^{2/3}N^{1/3}$. Notice that P is the number of qubits used here, while M is the amount of classical memory, and M is bigger.

Depth Optimization. Let us keep a local 2-dimensional grid with $S_G = P^{3/2}$ and a corresponding depth $\mathsf{H}_G^{4/5}N^{2/5}P^{-7/10}$. 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:

$$\mathsf{D}P = \mathcal{O}\left(\mathsf{H}_G^{8/7}N^{4/7}\mathsf{D}^{-3/7}\right)$$

which is valid as long as $P \leq N^{1/3}\mathsf{H}_G^{2/3}$ i.e. $N^{1/6}\mathsf{H}_G^{1/3} \leq \mathsf{D}$, and as long as $1 \leq P$ i.e. $\mathsf{D} \leq N^{2/5}\mathsf{H}_G^{4/5}$.

Further Parallelization To reach depths below $\widetilde{\mathcal{O}}(N^{1/6})$, we can parallelize by splitting the Grover search. With P_1 machines, each with M words of classical memory and P processors, we will only need $\sqrt{N\theta/MPP_1}$ Grover iterations for each machine. The depth is then

$$\mathcal{O}\left(\frac{M}{P}\frac{\mathsf{H}_D}{\sqrt{\theta}} + \sqrt{\frac{N\theta}{MPP_1}}\left(\frac{\mathsf{H}_D}{\sqrt{\theta}} + \frac{M}{P}S_D\right)\right) \tag{26}$$

and the gate count is

$$\mathcal{O}\left(P_1\left(M\frac{\mathsf{H}_G}{\sqrt{\theta}} + P\sqrt{\frac{N\theta}{MPP_1}}\left(\frac{\mathsf{H}_G}{\sqrt{\theta}} + \frac{M}{P^2}S_G\right)\right)\right) . \tag{27}$$

In this setting, the parameters with the lowest gate count are $M = P = N^{1/3}H_G^{2/3}P_1^{-1/3}$ and $\theta = H_G/P^{1/2}$. This leads to approximately 1 Grover iteration, so in fact only the search for distinguished points needs to be quantum.

To fit a depth limit D, we set $P_1 = \frac{N}{\mathsf{D}^6}\max\{\mathsf{H}_D^6/\mathsf{H}_G^4, \mathsf{H}_G^2\}$ for a total gate count of

$$\mathcal{O}\left(\frac{N}{\mathsf{D}^3}\max\left\{\frac{\mathsf{H}_D^3}{\mathsf{H}_G^2}, \mathsf{H}_G\right\}\right). \tag{28}$$

6 Security of SIKE

Supersingular Isogeny Key Encapsulation (SIKE) [JAC+17] is a candidate post-quantum key encapsulation based on isogenies of elliptic curves. So far generic meet-in-the-middle attacks outperform the best algebraic attacks, so its security is based on the difficulty of these 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 proposed security levels relative to the difficulty of attacks on symmetric cryptography, and separately imposed limitations on the total sequential operations an algorithm may perform, the "Maxdepth". SIKE-434, SIKE-503, SIKE-610, and SIKE-751 target NIST's security levels 1, 2, 3, and 5, respectively.

Levels 1, 3, and 5 are defined relative to key search on the AES block cipher. NIST used gate counts from Grassl et al. [GLRS16], but we use improved numbers from subsequent work [JNRV20]. They are given in Table 3. Levels 2 and 4 are based on collisions for the SHA family of hash functions. We use the collision search of [CNS17] and the results of Section 5. Table 3 shows the resulting costs when applied to SHA3 under NIST's depth restrictions.

Table 3: Security thresholds from NIST. AES key search figures are from [JNRV20]. The cost of evaluating SHA-3 is taken from [AMG+16].

AES key search					SHA Collisions			
		Security Level					Security Level	
Metric	Maxdepth	1	3	5	Metric	Maxdepth	2	4
G-cost	∞	83	116	148	G-cost	∞	122	184
	2^{96}	83	126	191		2^{96}	134	221
	2^{64}	93	157	222		2^{64}	147	267*
	2^{40}	117	181	246*		2^{40}	161	339*
DW-cost	∞	87	119	152	DW-cost	∞	134	201
	2^{96}	87	130	194		2^{96}	145	239
	2^{64}	97	161	225		2^{64}	159	286*
	2^{40}	121	185	249*		2^{40}	186	357*
Classical		143	207	272	Classical		146	210

Resource restrictions. NIST restricts the total circuit depth available to an attack because a real-world adversary will have some time limits. For example, if a secret is only valuable for 10 years, then it does not matter how little an algorithm costs if it takes longer than that.

Unlike many classical algorithms, most quantum algorithms parallelize very poorly so a depth limit forces very large hardware requirements. NIST does not give any fixed limit on the total hardware available. Providing a reasonable and compelling upper bound on the number of qubits is beyond the scope of this work. Instead we will choose 2^{200} as a loose guess, which is approximately the number of baryons in the solar system. Any cost that requires more than 2^{200} qubits in this section is marked with a $*$.

Security estimates. Because of the depth restriction, we focus on the parallel prefix-based walk and parallel Multi-Grover.

To find non-asymptotic estimates, we ignore many constant factors. For example, the depth of a 2-dimensional mesh sorting network of R elements is likely closer to $3R^{1/2}$ [Kun87]. We also need estimates of the cost of H, and we use those from [JS19]. Overall our results are likely to underestimate the real cost by constant or poly-logarithmic factors.

In the massively parallel parameterizations, once each processor has finished, we must assemble the results. This is an easy check, but if the total hardware is too large, the time for the signals to propagate exceeds the maximum depth. We ignore this restriction, though this should be considered when interpreting our results for extremely large hardware.

Table 4 shows the costs to attack various SIKE parameters under different depth restrictions, and shows by how many bits 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 [JS19], but not enough to reduce the claimed security levels. Because neither algorithm can parallelize well, both must resort to Grover-like parallelizations and this leads to high costs.

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$\times$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.

If we ignored locality issues, then the Multi-Grover algorithm could parallelize 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 assumptions about 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 over the gate counts and depth-width products of previous algorithms when cheap "qRAM" operations are not available. This study showed that, in this model, the NIST candidate SIKE offers less security than claimed in [JS19], but still more than the initial levels given in [JAC+17].

Using two different techniques, we arrived at a gate complexity of $\mathcal{O}(N^{6/7})$ 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 [Amb07]: $T^2 \times R = N^2$, so we did not obtain an improvement in depth$\times$width. However, our result can be reformulated more positively: qRAM is not necessary if we use less than $N^{2/7}$ memory. A similar situation occurred in the context of (non-golden) collision search, where [BHT98] and [CNS17] achieve the same tradeoff curve $T^2 \times R = N$, although the second one only goes as far as $T = N^{2/5}$.

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 partially supported by the University of Oxford Clarendon fund, and would like to thank Christophe Petit and Richard Meister for helpful comments.

References

ACVCD+. G. Adj, D. Cervantes-Vázquez, J.-J. Chi-Domínguez, A. Menezes, and F. Rodríguez-Henríquez. On the cost of computing isogenies between supersingular elliptic curves. In *Selected Areas in Cryptography – SAC 2018*, pages 322–343. LNCS 11349.

AGJO+15. S. Arunachalam, V. Gheorghiu, T. Jochym-O'Connor, M. Mosca, and P. V. Srinivasan. On the robustness of bucket brigade quantum ram. *New Journal of Physics*, 17(12):123010, 2015.

Amb07. A. Ambainis. Quantum walk algorithm for element distinctness. *SIAM J. Computing*, 37:210–239, 2007.

AMG+16. M. Amy, O. Di Matteo, V. Gheorghiu, M. Mosca, A. Parent, and J. M. Schanck. Estimating the cost of generic quantum pre-image attacks on SHA-2 and SHA-3. In *SAC 2016*, pages 317–337, 2016.

APS15. M. Albrecht, R. Player, and S. Scott. On the concrete hardness of Learning with Errors. *Journal of Mathematical Cryptology*, pages 169–203, 2015.

AS04. S. Aaronson and Y. Shi. Quantum lower bounds for the collision and the element distinctness problems. *J. ACM*, 51(4):595–605, July 2004.

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. "Security margin" is the difference between the cost of the cheapest *local* quantum attack (**in bold**) and the cheapest quantum or classical attack from Table 3.

Nonlocal Prefix-based walk					
		SIKE p bitlength			
Metric	Depth	434	503	610	751
G	∞	109	125	148	179
	2^{96}	111	132	165	208
	2^{64}	124	145	179	235
	2^{40}	136	166	220	291*
DW	∞	148	168	200	241
	2^{96}	147	161	183	212
	2^{64}	134	148	179	235
	2^{40}	136	166	220	291*

Local Prefix-based walk					
		SIKE p bitlength			
Metric	Depth	434	503	610	751
G	∞	**109**	**125**	**148**	**179**
	2^{96}	**111**	**135**	185	257
	2^{64}	147	183	237	309*
	2^{40}	186	221	276*	347*
DW	∞	148	168	200	241
	2^{96}	147	168	221	293
	2^{64}	173	208	263	334*
	2^{40}	204	239	294*	366*

Local Multi-Grover					
		SIKE p bitlength			
Metric	Depth	434	503	610	751
G	∞	131	149	177	213
	2^{96}	131	149	**181**	**235**
	2^{64}	**145**	**172**	**213**	**268**
	2^{40}	**169**	**196**	**237***	**292***
DW	∞	**141**	**159**	**187**	**223**
	2^{96}	**141**	**159**	**191**	**246**
	2^{64}	**155**	**182**	**223**	**279**
	2^{40}	**179**	**206**	**247***	**302***

Security Margin					
		SIKE p bitlength			
Metric	Depth	434	503	610	751
G	∞	27	3	32	31
	2^{96}	28	1	55	44
	2^{64}	52	26	56	46
	2^{40}	52	50	56*	46*
DW	∞	62	35	81	89
	2^{96}	54	14	61	52
	2^{64}	58	36	62	54
	2^{40}	58	60	62*	53*

BB18. Gustavo Banegas and Daniel J. Bernstein. Low-Communication Parallel Quantum Multi-Target Preimage Search. In Carlisle Adams and Jan Camenisch, editors, *Selected Areas in Cryptography – SAC 2017*, volume 10719, pages 325–335. Springer International Publishing, Cham, 2018.

BBG$^+$13. R. Beals, S. Brierley, O. Gray, A. W. Harrow, S. Kutin, N. Linden, D. Shepherd, and M. Stather. Efficient distributed quantum computing. *Proc. Royal Soc. London A: Mathematical, Physical and Engineering Sciences*, 469, 2013.

BDH$^+$05. H. Buhrman, C. Dürr, M. Heiligman, P. Høyer, F. Magniez, M. Santha, and R. de Wolf. Quantum algorithms for element distinctness. *SIAM J. Comput.*, 34(6):1324–1330, 2005.

Ben89. C. H. Bennett. Time/space trade-offs for reversible computation. *SIAM J. Comput.*, 18(4):766–776, 1989.

Ber09. D. J. Bernstein. Cost analysis of hash collisions: Will quantum computers make SHARCS obsolete? *Workshop Record of SHARCS 2009: Special-purpose Hardware for Attacking Cryptographic Systems*, 2009.

BHMT02. G. Brassard, P. Hoyer, M. Mosca, and A. Tapp. Quantum amplitude amplification and estimation. *Contemporary Mathematics*, 305:53–74, 2002.

BHT98. G. Brassard, P. Høyer, and A. Tapp. Quantum cryptanalysis of hash and claw-free functions. In *Theoretical Informatics, Latin American Sympsium – LATIN 1998*, LNCS 1380, pages 163–169, 1998.

CLN$^+$. C. Costello, P. Longa, M. Naehrig, J. Renes, and F. Virdia. Improved classical cryptanalysis of sike in practice. In *Public Key Cryptography – PKC 2020*, LNCS 12111.

CNS17. A. Chailloux, M. Naya-Plasencia, and A. Schrottenloher. An efficient quantum collision search algorithm and implications on symmetric cryptography. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017*, LNCS 10625, pages 211–240. Springer, 2017.

FO90. Philippe Flajolet and Andrew M. Odlyzko. Random Mapping Statistics. In Jean-Jacques Quisquater and Joos Vandewalle, editors, *Advances in Cryptology — EUROCRYPT '89*, pages 329–354, Berlin, Heidelberg, 1990. Springer Berlin Heidelberg.

GLM08. V. Giovannetti, S. Lloyd, and L. Maccone. Architectures for a quantum random access memory. *Physical Review A*, 78(5):052310, 2008.

GLRS16. M. Grassl, B. Langenberg, M. Roetteler, and R. Steinwandt. Applying grover's algorithm to AES: quantum resource estimates. In Tsuyoshi Takagi, editor, *PQCrypto 2016*, pages 29–43. Springer, 2016.

GR04. L. K. Grover and T. Rudolph. How significant are the known collision and element distinctness quantum algorithms? *Quantum Information & Computation*, 4(3):201–206, 2004.

Gro96. L. K. Grover. A Fast Quantum Mechanical Algorithm for Database Search. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing 1996*, pages 212–219. ACM, 1996.

JAC$^+$17. D. Jao, R. Azarderakhsh, M. Campagna, C. Costello, L. De Feo, B. Hess, A. Jalali, B. Koziel, B. LaMacchia, P. Longa, M. Naehrig, J. Renes, V. Soukharev, and D. Urbanik. Supersingular isogeny key encapsulation. *Submission to NIST post-quantum project*, November 2017. available at `https://sike.org/#nist-submission`.

JNRV20. S. Jaques, M. Naehrig, M. Roetteler, and F. Virdia. Implementing grover oracles for quantum key search on AES and LowMC. In *EUROCRYPT 2020*, LNCS 12106, 2020.

JS19. S. Jaques and J. M. Schanck. Quantum cryptanalysis in the RAM model: Claw-finding attacks on SIKE. In *CRYPTO 2019*, LNCS 11693, pages 32–61, 2019.

Kun87. M. Kunde. Lower bounds for sorting on mesh-connected architectures. *Acta Inf.*, 24(2):121–130, 1987.

Kup13. G. Kuperberg. Another subexponential-time quantum algorithm for the dihedral hidden subgroup problem. In *TQC 2013*, LIPIcs 22, pages 20–34, 2013.

LS90. R. Y. Levin and A. T. Sherman. A note on Bennett's time-space tradeoff for reversible computation. *SIAM J. Comput.*, 19(4):673–677, 1990.

MNRS11. F. Magniez, A. Nayak, J. Roland, and M. Santha. Search via quantum walk. *SIAM Journal on Computing*, 40:142–164, 2011.

NC02. M. A. Nielsen and I. Chuang. *Quantum computation and quantum information.* AAPT, 2002.

NIS16. NIST. Submission requirements and evaluation criteria for the post-quantum cryptography standardization process, 2016.

NS20. M. Naya-Plasencia and A. Schrottenloher. Optimal Merging in Quantum k-xor and k-sum Algorithms. In *EUROCRYPT 2020*, LNCS 12106, 2020.

RS67. A. Rényi and G. Szekeres. On the height of trees. *Journal of the Australian Mathematical Society*, 7(4):497–507, November 1967.

Tan09. S. Tani. An improved claw finding algorithm using quantum walk. In *Mathematical Foundations of Computer Science – MFCS 2007*, pages 548–558. LNCS 4708, 2009.

vOW99. P.C. van Oorschot and M.J. Wiener. Parallel collision search with cryptanalytic applications. *J.Cryptology*, 12(1):1–28, Jan 1999.

A Quantum Circuits for Iterations

In this section, we give details on 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. The set-up can be accomplished by sequential insertion steps (which are part of the update), and we will maintain a counter or flag for each list indicating whether it is marked, giving unit cost to the check step. Thus, the main analysis is the update step. We use the Johnson vertex data structure from [JS19]. 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 $(x, h(x), h^2(x), \dots, h^{n_x}(x))$, where $h^{n_x}(x)$ is distinguished. The goal of this sub-circuit 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 be guaranteed to reach a distinguished point.

The length of trails is geometrically distributed [vOW99], with a mean equal to $1/\theta$ if the fraction of distinguished points is θ. Let n be the number of iterations we choose. The proportion of trails with length greater than $n = c/\theta$ is approximately e^{-c} [vOW99].

Pebbling. Since h is by definition non-injective, it cannot be applied in-place, so we will need a pebbling strategy (see *e.g.* [Ben89,LS90]). The same issue is described in [BB18]. 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. We want to output the last distinguished point that h reaches. To do this, we will 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 the $i+1$ location, and the last element is moved to the beginning. Then the output is copied to the first element in the list.

As long as the *total* number of distinguished points reached is less than k, 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 a reduction in the number of marked vertices.

Error Analysis. There are two sources of error: trails that do not find any distinguished point in n collisions, and trails that find more than k distinguished points. For $u = c/\theta$, the probability of the first error is e^{-c} and the probability of the second is at most $\frac{c}{k}$, by Markov's inequality. Thus, we can take $u = \mathcal{O}(1/\theta)$ and $k = \mathcal{O}(1)$. If we further assume that the number of distinguished points in a trail follows a binomial distribution, then the probability of too many distinguished points is much smaller.

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[4]. 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, 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.

[4] 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.

These loops do not change the time complexity at all, and it easy to create such a loop 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.

A.2 Finding Collisions

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

This makes our collision-finding circuit substantially simpler. We can slightly modify the search circuit on a Johnson vertex [JS19] to do this. 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 simply 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 or not.

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 shows that this ensures that if we only have one predecessor from each half of the golden collision, the vertex will be marked, and that the only marked vertices will be 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. Ultimately, 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 will have 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.

A.5 Constant Savings

Note that there are many points here where we could reduce the cost by not uncomputing intermediate iterations of h applied to the new point. We ignore such optimizations, since they only give a constant factor improvement.

B Probability Analysis

The analysis of van Oorschot and Wiener [vOW99] 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 [FO90] for other standard results).

We define the set of predecessors of x as

$$\mathcal{P}_x = \{y \in X \,|h^n(y) = x, n \geq 0\}\,. \tag{29}$$

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*

$$\Pr[P_x = t] = \frac{t^{t-1}}{e^t t!}\left(1 + \mathcal{O}(\tfrac{1}{N})\right) \tag{30}$$

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

Proof. We count the number of such functions. We select $t-1$ elements out of the $N-1$ elements which are not x to be x's predecessors. 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

$$\frac{\binom{N-1}{t-1}t^{t-2}N(N-t)^{N-t}}{N^N} = \frac{t^{t-1}}{t!}\frac{N!}{N^N}\frac{(N-t)^{N-t}}{(N-t)!}. \tag{31}$$

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

$$\frac{t^{t-1}e^{-t}}{t!}\sqrt{\frac{N}{N-t}}\left(1+\mathcal{O}(\tfrac{1}{N})\right). \tag{32}$$

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 $t, s = o(N)$,*

$$\Pr[P_x = t, P_y = s] = \frac{t^{t-1}s^{s-1}}{e^t t! e^s s!}\left(1+\mathcal{O}(\tfrac{1}{N})\right) \tag{33}$$

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

Proof. We assume $s \geq t$ without loss of generality.

First note that 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 $h(x) = h(y)$. Then either their sets of predecessors are disjoint, or x is a predecessor of y (meaning $h(x)$ is a predecessor of y). We cannot have y as a predecessor of x, because then x would have more predessors than y, contradicting our assumption.

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

$$\frac{\binom{N-2}{t-1}t^{t-2}\binom{N-t-1}{s-1}s^{s-2}(N-t-s)^{N-t-s}(N-t-s)}{N^{N-1}}. \tag{34}$$

This can be simplified and then approximated to

$$\frac{t^{t-1}}{t!}\frac{s^{s-1}}{s!}\frac{N!}{N^N}\frac{(N-t-s)^{N-t-s}}{(N-t-s)!}\frac{N-t-s}{N-1} = \frac{t^{t-1}}{e^t t!}\frac{s^{s-1}}{e^s s!}(1+\mathcal{O}(\tfrac{1}{N})). \tag{35}$$

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

$$\frac{\binom{N-2}{s-2}\binom{s-2}{t-1}t^{t-2}(s-t)^{s-t-2}(s-t)(N-s)^{N-s}}{N^{N-1}}. \tag{36}$$

This can be simplified to

$$\frac{t^{t-1}}{t!}\frac{(s-t)^{s-t}}{(s-t)!}\frac{N!}{N^N}\frac{(N-s)^{N-s}}{(N-s)!}\frac{1}{N-1} \tag{37}$$

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

$$\frac{1}{N}\left(\frac{t^{t-1}}{e^t t!}\frac{(s-t)^{s-t}}{e^{s-t}(s-t)!}\right) \tag{38}$$

which fits within the error term of Equation 35, 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*

$$\frac{2(\pi-3)}{3(c-1)^2}\left(1+\mathcal{O}(\tfrac{s}{N})\right). \tag{39}$$

Proof. The probability that a tree on n vertices has maximum height h will decrease with n [RS67], so we can assume that x and y have disjoint trees of predecessors and this will overestimate the probability of tall predecessor trees.

By [RS67], 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 precessors with small height. Recall that 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*

$$\Theta\left(\frac{R^2ts}{N^2}\right). \tag{40}$$

Proof. Define the u-predecessors of x by

$$\mathcal{P}_u(x) = \{p \in X \,|h^m(p) = x, u \geq m \geq 0\}. \tag{41}$$

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 just t and s. This acts as a multinomial distribution, and thus the probability of one element from each set is

$$\binom{R}{2}\frac{t}{N}\frac{s}{N}\left(1-\frac{t+s}{N}\right)^{R-2} = \Theta\left(\frac{R^2ts}{N^2}\right). \tag{42}$$

□

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*

$$\frac{\pi-3}{3(c-1)^2}. \tag{43}$$

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. Consider choosing a random subset of m of these predecessors. If we 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 trees formed from the remaining $t-m$ points. As long as we do not connect any two of the m points, we will not form a tree. Any such arrangement of trees and connections produces a valid and distinct tree out of the 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 [RS67], the probability of this tree having height greater than u is at most

$$\frac{(\pi - 3)}{3(c-1)^2}. \tag{44}$$

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. It's possible that the number of close predecessors grows 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*

$$\frac{2(\pi - 3)}{2(c-1)^2}\left(1 + \mathcal{O}(\tfrac{s}{N})\right) \tag{45}$$

the fraction of marked vertices, when iterating h at least u times, is

$$\Omega\left(\frac{R^2 \min\{u^2, t\} \min\{u^2, s\}}{N^2}\right) \tag{46}$$

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.

The same reasoning holds for y. Thus, by the same reasoning as Theorem B.1, we have the result. 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

$$(1-\theta)^{u_2} < e^{-\theta u_2} \tag{47}$$

and thus $u_2 = \Omega(1/\theta)$ will give at least 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. $\square$