Deterministic Algorithms for the Hidden Subgroup Problem

Ashwin Nayak *
University of Waterloo

March 15, 2022

Abstract

We present deterministic algorithms for the Hidden Subgroup Problem. The first algorithm, for abelian groups, achieves the same asymptotic worst-case query complexity as the optimal randomized algorithm, namely $O(\sqrt{n})$, where n is the order of the group. The analogous algorithm for non-abelian groups comes within a $\sqrt{\log n}$ factor of the optimal randomized query complexity.

The best known randomized algorithm for the Hidden Subgroup Problem has *expected* query complexity that is sensitive to the input, namely $O(\sqrt{n/m})$, where m is the order of the hidden subgroup. In the first version of this article [10, Sec. 5], we asked if there is a deterministic algorithm whose query complexity has a similar dependence on the order of the hidden subgroup. Prompted by this question, Ye and Li [14] present deterministic algorithms for *abelian* groups which solve the problem with $O(\sqrt{n/m})$ queries, and find the hidden subgroup with $O(\sqrt{n(\log m)/m} + \log m)$ queries. Moreover, they exhibit instances which show that in general, the deterministic query complexity of the problem may be $o(\sqrt{n/m})$, and that of *finding* the entire subgroup may also be $o(\sqrt{n/m})$ or even $\omega(\sqrt{n/m})$.

We present a different deterministic algorithm for the Hidden Subgroup Problem that also has query complexity $O(\sqrt{n/m})$ for abelian groups. The algorithm is arguably simpler. Moreover, it works for non-abelian groups, and has query complexity $O(\sqrt{(n/m)\log(n/m)})$ for a large class of instances, such as those over supersolvable groups. We build on this to design deterministic algorithms to find the hidden subgroup for all abelian and some non-abelian instances, at the cost of a $\log m$ multiplicative factor increase in the query complexity.

1 Introduction

In the Simon Problem with parameter k, we are given an oracle for a function $f : \mathbb{Z}_2^k \to S$, for some co-domain S with $|S| \geq 2^k$. The function f is either injective, or there is an unknown non-zero element $s \in \mathbb{Z}_2^k$ such that for all $x, y \in \mathbb{Z}_2^k$, we have $f(x) = f(y)$ if and only if $x + y \in \{0, s\}$. In the latter case, we say that the function "hides" the element s, and call a pair of distinct inputs x, y a "collision" if $f(x) = f(y)$. The task is to determine which of the two cases holds.

Simon designed a bounded-error quantum algorithm with query complexity $O(k)$ for the eponymous problem, and showed that any bounded-error classical (i.e., randomized) algorithm for the problem requires $\Omega(2^{k/2})$ queries [11]. (The query lower bound stated in the paper is $\Omega(2^{k/4})$ for classical algorithms with error at most $\frac{1}{2} - 2^{-k/2}$. However, the proof can be modified in a straightforward manner to show that if the algorithm makes error at most $\frac{1}{4}$, at least $2^{k/2-1}$

*Department of Combinatorics and Optimization, and Institute for Quantum Computing, University of Waterloo, 200 University Ave. W., Waterloo, ON, N2L 3G1, Canada. Email: ashwin.nayak@uwaterloo.ca .

queries are required.) The lower bound for classical algorithms is optimal up to a constant factor, as is implied by the "Birthday Paradox": if we pick $t := \lceil 2^{k/2+1} \rceil$ elements $X_1, X_2, \ldots, X_t$ independently and uniformly at random from $\mathbb{Z}_2^k$, if f hides some non-zero element, with probability at least $3/4$ we find a collision (i.e., a pair X_i, X_j with $i, j \in [t]$ such that $X_i \neq X_j$ but $f(X_i) = f(X_j)$).

We present a simple, *deterministic* algorithm for the Simon problem that achieves the asymptotically optimal classical query complexity (Theorem 2.1). Since posting the first version of this article [10], we have learnt that algorithms achieving the same asymptotic query complexity, including the same algorithm, were known before [5, 12, 13]. Nevertheless, it is instructive to understand the algorithm underlying Theorem 2.1, as it forms the basis of the generalizations that we describe next.

The Simon Problem is a special case of the Hidden Subgroup Problem (see, e.g., Ref. [6]). In the Hidden Subgroup Problem, we are given the description of a finite group G, and an oracle for a function $f : G \to S$, where $|S| \geq |G|$. The function f is either injective, or there is an unknown non-trivial subgroup H of G such that for all $x, y \in G$, we have $f(x) = f(y)$ if and only if $x^{-1}y \in H$. In other words, the function f is constant on left cosets of a possibly trivial subgroup of G, and takes distinct values for distinct left cosets of the subgroup. We say that the function "hides" the subgroup H. The task is to determine whether the subgroup is trivial, i.e., the function f is injective, or not. When $G := \mathbb{Z}_2^k$ and the hidden subgroup H is of the form $\{0, s\}$ for some unknown non-zero element $s \in \mathbb{Z}_2^k$, we get the Simon Problem.

The randomized algorithm for the Simon Problem generalizes immediately to the Hidden Subgroup Problem when the order m of the hidden subgroup H is known, and the resulting algorithm has query complexity $O(\sqrt{n/m})$, where n is the order of the group G. When the order m is not known, we can use this basic algorithm to design a new algorithm. The new algorithm has query complexity $O(\sqrt{n})$ when the function is injective, and *expected* query complexity $O(\sqrt{n/m})$ when the function f hides some non-trivial subgroup H with unknown order m. Namely, we run the basic randomized algorithm above, with error at most $1/4$, assuming that the order of the hidden subgroup is at least r, starting from $r := n/2$. If we do not succeed, we halve r, and repeat (until $r \leq 1$).

We may ask if the Hidden Subgroup Problem also admits a deterministic algorithm that is as efficient in the worst-case as the best randomized algorithm. We answer this in the affirmative when the underlying group is abelian (Corollary 3.6). For non-abelian groups, we present a deterministic algorithm that comes within a multiplicative factor of $O(\sqrt{\log n})$ of the optimal randomized query complexity (Corollary 4.2). Both the algorithms are based on the construction of a *generating pair* of subsets for a group (Definition 3.1), with optimal or nearly optimal size.

The optimal randomized algorithm for the Hidden Subgroup Problem described above has expected query complexity that is sensitive to the input, namely $O(\sqrt{n/m})$. It is natural to ask if there is a deterministic algorithm that has query complexity with a similar dependence on the order of the hidden subgroup. Algorithms coming close to this bound were known prior to this work for the case of $G := \mathbb{Z}_p^k$ for prime p, when the subgroup H has order p^l and l is known; see Ref. [12] for the $p = 2$ case, and Ref. [13] for the general case. The algorithm in Ref. [13] has query complexity $O(\sqrt{n(\log m)/m} + \log m)$.

The above question was posed as an open problem in the first version of this article [10, Sec. 5]. Prompted by this question, Ye and Li [14] present deterministic algorithms for *abelian* groups which solve the problem with query complexity $O(\sqrt{n/m})$, and find the hidden subgroup with query complexity $O(\sqrt{n(\log m)/m} + \log m)$. These algorithms also build on the concept of generating pairs which we introduced in Ref. [10], and use a construction of generating pairs similar to the one we gave. Moreover, Ye and Li exhibit instances which show that in general, the deterministic query complexity of the problem may be $o(\sqrt{n/m})$, and that of *finding* the entire subgroup may

also be $o(\sqrt{n/m})$ or even $\omega(\sqrt{n/m})$. In fact, the instance with complexity $\omega(\sqrt{n/m})$ for finding the hidden subgroup follows from earlier work due to Ye, Huang, Li, and Wang [13, Theorem 1]. In more detail, for the group $\mathbb{Z}_{p^k}$ for a given prime p and $k \geq 2$, the deterministic query complexity of the problem is 2 and that of finding the hidden subgroup is $O(\log(n/m))$, while n/m may be $\omega(1)$. For $G := \mathbb{Z}_p^k$ and a hidden subgroup H of order p^{k-1}, any deterministic algorithm that *finds* the subgroup has query complexity $\Omega(k)$, while $\sqrt{n/m}$ is $\sqrt{p}$. This gives an arbitrarily large separation as k grows for a fixed prime p.

We present a different deterministic algorithm for the Hidden Subgroup Problem that also has query complexity $O(\sqrt{n/m})$ for abelian groups (Algorithm 1, Theorem 5.2). The algorithm is arguably simpler. Furthermore, it also works for non-abelian groups, and has query complexity $O(\sqrt{(n/m)\log(n/m)})$ for classes of instances which include those over supersolvable groups. We observe that for abelian groups the problem of *finding* the hidden subgroup may be reduced to that of finding a single collision, and obtain an $O\big((\log m)\sqrt{n/m}\big)$-query deterministic algorithm for it (Algorithm 2, Theorem 6.1). We build on these results to design an algorithm that finds the hidden subgroup in certain non-abelian instances with $O\big((\log m)\sqrt{(n/m)\log(n/m)}\big)$ queries (Algorithm 3, Theorem 6.3). The instances are precisely the ones in which the underlying group is a bicrossed product of the hidden subgroup with another subgroup [1]. All these algorithms again rest on generating pairs of (near-) optimal size.

Note that the query complexity of the algorithm due to Ye and Li for abelian groups may be a factor of $\sqrt{\log m}$ smaller than that of Algorithm 2. They achieve the stronger bound by searching for a structured set of independent generators for the hidden subgroup, and by using a partially nested sequence of generating pairs with finely tuned size. This entails a detailed analysis of the structure of the hidden subgroup. It is not clear whether we can achieve the same query complexity without resorting to these ideas.

Acknowledgements. A.N. is grateful for the opportunity to teach quantum computation at the undergraduate level, which prompted this work. He is also grateful to Kanstantsin Pashkovich for several helpful discussions, especially for a course-correction early in this work. He thanks William Slofstra for a pointer to relevant literature, and Zekun Ye for bringing Ref. [13] to his attention. This research is supported in part by a Discovery Grant from NSERC Canada.

2 The Simon Problem

We start with a simple, deterministic algorithm for the Simon problem that matches the query complexity of the asymptotically optimal classical algorithm.

Theorem 2.1. *There is a deterministic algorithm that makes $2^{\lfloor k/2 \rfloor} + 2^{\lceil k/2 \rceil}$ queries and solves the Simon Problem with parameter k. Moreover, if the input function hides the non-zero element s, the algorithm finds s. Finally, any deterministic algorithm for the problem requires at least q queries, where q is the smallest positive integer such that $\binom{q}{2} \geq 2^k - 1$.*

Proof. Let $l := \lfloor k/2 \rfloor$. Viewing elements of $\mathbb{Z}_2^k$ as k-bit strings, we query the function f at all 2^l elements of the form $u\,0^{k-l}$, where $u \in \mathbb{Z}_2^l$, and at all 2^{k-l} elements of the form $0^l\,v$, where $v \in \mathbb{Z}_2^{k-l}$. If the function is distinct at all these points, we say f is injective. Otherwise, we say that f is not injective, and output $x + y$, where $x \neq y$ and x, y are a colliding pair (i.e., are such that $f(x) = f(y)$).

If the function is injective, the above algorithm outputs the correct answer. Suppose f hides a non-zero element $s \in \mathbb{Z}_2^k$. Let a be the projection of s onto the first l coordinates, and b be the

projection of s onto the last $k-l$ coordinates. Then $a\,0^{k-l} + 0^l\,b = s$. As $f(a\,0^{k-l}) = f(0^l\,b)$, the above algorithm detects a collision and computes s correctly.

Now consider any deterministic algorithm for the problem that makes t queries such that

$$\binom{t}{2} \quad \leq \quad 2^k - 2 \ .$$

Then we argue that there is an injective function f_0, and a function f_1 that hides a non-zero element s, such that f_0 and f_1 agree on all the t elements queried. Suppose the t queries that the algorithm makes are $x_1, x_2, \ldots, x_t$. We take f_0 to be any injective function. Consider the set of elements $S := \{x_i + x_j : i, j \in [t], i \neq j\}$. We have $|S| \leq \binom{t}{2} \leq 2^k - 2$, so there is at least one non-zero element in $\mathbb{Z}_2^k \setminus S$. Let s be one such element. By definition of s, we have $x_i + s \neq x_j$ for any $i, j \in [t]$. Thus there is a function f_1 that equals f_0 at all the points x_i, $i \in [t]$, and also hides s. □

Note that for $k \geq 2$, if $\binom{q}{2} \geq 2^k - 1$, we have $q^2 \geq 2^{k+1} - 2$, and therefore

$$q \quad \geq \quad 2^{(k+1)/2}(1 - 1/2^k)^{1/2} \quad \geq \quad 2^{(k+1)/2} - 2^{-(k-1)/2} \ .$$

Since q is integral, $q \geq 2^{(k+1)/2}$. The deterministic upper bound in the theorem comes within a factor of $3/2$ of this lower bound. Also note that the upper bound in the theorem is within a factor of $3\sqrt{2}$ of the query lower bound for randomized algorithms with error at most $1/4$.

3 The Abelian Hidden Subgroup Problem

We now turn our focus to the Hidden Subgroup Problem when the underlying group is abelian. It is not clear how the use of projections in the algorithm presented in Section 2 may be extended to this case, let alone to the non-abelian case. The issue is that the given abelian group may be the direct product of cyclic groups with vastly different orders. We give a different extension of the algorithm, which combines several ways of expressing an abelian group as a sum of two subsets.

More formally, the basic idea behind the algorithm in Theorem 2.1 is to find a pair of sets S_1, S_2 of elements of the group G such that $|S_1|, |S_2| \in O(\sqrt{|G|})$ and $S_1 + S_2 = G$, where

$$S_1 + S_2 \quad := \quad \{x + y : x \in S_1,\ y \in S_2\} \ .$$

(Here, '+' denotes the group operation.) We explain how such a pair of sets may be constructed in a few cases. Together, they yield a construction for a general abelian group.

Definition 3.1. *For any group G, we say a pair $S_1, S_2 \subseteq G$ is a* generating pair *for G if the set $S_1 S_2$ defined as $S_1 S_2 := \{xy : x \in S_1,\ y \in S_2\}$ equals G.*

We start with the construction of a generating pair for a cyclic group.

Lemma 3.2. *Let n be an integer greater than 1, and let $m := \lceil \sqrt{n} \rceil$. There is a generating pair S_1, S_2 for $\mathbb{Z}_n$ such that $|S_1| = m$ and $|S_2| \leq \lfloor n/m \rfloor + 1$. If n is a perfect square, the pair further satisfies $|S_1| = |S_2| = \sqrt{n}$.*

Proof. We use the Division Algorithm to find such subsets. If we choose a divisor m that roughly equals $\sqrt{n}$, then the number of different remainders and quotients we get when we divide non-negative integers less than n are both roughly $\sqrt{n}$.

Let $m := \lceil \sqrt{n} \rceil$, let $S_1 := \{0, 1, 2, \ldots, m-1\}$, and let $S_2 := \{mi : 0 \leq i < n/m\}$. The subsets S_1, S_2 satisfy the required properties. □

Next, we consider a direct product of two cyclic groups whose orders are both odd powers of the same prime number.

Lemma 3.3. *Consider the group $G := \mathbb{Z}_n \times \mathbb{Z}_m$, where $n := p^k$, $m := p^l$, p is prime, and k, l are positive odd integers. There is a generating pair S_1, S_2 for G such that $|S_1| = |S_2| = \sqrt{nm}$.*

Proof. W.l.o.g., assume that $k \geq l$. Let $q := p^{(k+l)/2}$. Note that $0 < q = \sqrt{nm} < n$. Consider

$$\begin{aligned} S_1 &:= \mathbb{Z}_q \times \{0\} \ , \quad \text{and} \\ S_2 &:= \{iq : 0 \leq i < n/q\} \times \mathbb{Z}_m \ . \end{aligned}$$

By the Division Algorithm, we have

$$\mathbb{Z}_q + \{iq : 0 \leq i < n/q\} \quad = \quad \mathbb{Z}_n \ .$$

So $S_1 + S_2 = G$. Moreover, we have $|S_1| = q = \sqrt{nm} = (n/q) \times m = |S_2|$. □

We may combine the generating pairs for two groups to obtain one for their direct product. While we present the proof of the lemma below with the notation for abelian groups, it also holds for non-abelian groups.

Lemma 3.4. *Consider a direct product group $G := G_1 \times G_2$. Suppose S_1, S_2 is a generating pair for G_1, and T_1, T_2 is a generating pair for G_2. Then $(S_1 \times T_1), (S_2 \times T_2)$ is a generating pair for G.*

Proof. Any element $g \in G_1$ may be expressed as $g_1 + g_2$, where $g_1 \in S_1$ and $g_2 \in S_2$. Similarly, any element $h \in G_2$ may be expressed as $h_1 + h_2$, where $h_1 \in T_1$ and $h_2 \in T_2$. Then $(g, h) = (g_1, h_1) + (g_2, h_2) \in (S_1 \times T_1) + (S_2 \times T_2)$. □

The construction for a general finite abelian group combines the above building blocks.

Theorem 3.5. *For any finite abelian group G with order n, there is a generating pair S_1, S_2 for G such that $|S_1|$ and $|S_2|$ are both at most $2\sqrt{n}$.*

Proof. By the fundamental theorem of abelian groups, any non-trivial finite abelian group G may be expressed as a direct product of cyclic groups of prime power order:

$$G \quad \cong \quad \mathbb{Z}_{p_1^{k_1}} \times \mathbb{Z}_{p_2^{k_2}} \times \cdots \times \mathbb{Z}_{p_l^{k_l}} \ , \tag{3.1}$$

where the integers p_i are primes, not necessarily distinct, and the integers $k_i \geq 1$ for all $i \in [l]$. We prove the lemma by strong induction on l, the number of cyclic groups in a decomposition of G.

If $l = 1$, the statement follows from Lemma 3.2.

Suppose the statement holds for all non-trivial finite abelian groups which have a decomposition as above with at most m cyclic groups, for some $m \geq 1$. Suppose $l := m + 1$, and consider a group G with order n and a decomposition with l cyclic groups as in eq. (3.1).

Suppose for some $i \in [l]$, the integer k_i is even. Let r be such an index. We let $G_1 := \mathbb{Z}_{p_r^{k_r}}$ and G_2 the direct product of the remaining cyclic groups so that $G \cong G_1 \times G_2$. By Lemma 3.2, there is a generating pair S_1, S_2 for G_1 such that $|S_1| = |S_2| = \sqrt{n_1}$, where $n_1 := {p_r}^{k_r}$. By the induction hypothesis, there is a generating pair T_1, T_2 for G_2 such that $|T_1|, |T_2| \leq 2\sqrt{n_2}$, where $n_2 := |G_2|$. By Lemma 3.4, we get a generating pair $(S_1 \times T_1), (S_2 \times T_2)$ for G such that $|S_1 \times T_1|, |S_2 \times T_2| \leq 2\sqrt{n_1 n_2} = 2\sqrt{n}$.

Suppose for all $i \in [l]$, the integers k_i are odd. Suppose for some $i, j \in [l]$, $i \neq j$, we have $p_i = p_j$. Let r, s be such a pair of indices. We let $G_1 := \mathbb{Z}_{p_r^{k_r}} \times \mathbb{Z}_{p_s^{k_s}}$ and G_2 the direct product of the

remaining cyclic groups so that $G \cong G_1 \times G_2$. By Lemma 3.3, there is a generating pair S_1, S_2 for G_1 such that $|S_1| = |S_2| = \sqrt{n_1}$, where $n_1 := {p_r}^{k_r} {p_s}^{k_s}$. By the induction hypothesis, there is a generating pair T_1, T_2 for G_2 such that $|T_1|, |T_2| \leq 2\sqrt{n_2}$, where $n_2 := |G_2|$. By Lemma 3.4, we get a generating pair $(S_1 \times T_1), (S_2 \times T_2)$ for G such that $|S_1 \times T_1|, |S_2 \times T_2| \leq 2\sqrt{n_1 n_2} = 2\sqrt{n}$.

Suppose for all $i \in [l]$, the integers k_i are odd, and the primes p_i are all distinct. Then, by the Chinese Remainder Theorem, we have $G \cong Z_n$. The statement now follows from Lemma 3.2, by noting that for $n \geq 2$ we have $\lceil \sqrt{n} \rceil \leq 2\sqrt{n}$ and $\lfloor \sqrt{n} \rfloor + 1 \leq 2\sqrt{n}$. □

The algorithm for the abelian Hidden Subgroup Problem follows directly from the existence of a suitable generating pair for the underlying group.

Corollary 3.6. *There is a deterministic algorithm with query complexity at most $4\sqrt{n}$ that solves the Hidden Subgroup Problem over an abelian group G with order n. Moreover, if the input function hides the non-trivial subgroup H, the algorithm finds all the elements of H.*

Proof. By Theorem 3.5, there is a generating pair S_1, S_2 for the group G such that $|S_1|, |S_2| \leq 2\sqrt{n}$. We query the oracle function f at $-x$ for all $x \in S_1$ and at all elements $y \in S_2$. If the function is injective on the set of points queried, we say f is injective. Otherwise, we say that f is not injective, and output $\{x + y : x \in S_1,\ y \in S_2,\ f(-x) = f(y)\}$.

If the function is injective, the above algorithm outputs the correct answer. Suppose f hides the non-trivial subgroup H. Let $h \in H$ be any element of the hidden subgroup. We have $h = h_1 + h_2$ for some $h_1 \in S_1$ and $h_2 \in S_2$. By definition of H we have $f(-h_1) = f(h_2)$. When h is not the identity, we have $-h_1 \neq h_2$ and the algorithm detects a collision. Moreover, the algorithm computes H correctly. □

Note that we can get upper bounds with better constants for certain abelian groups by appealing to the properties of the generating pair constructed in Theorem 3.5. As the Simon Problem is a special case, Theorem 2.1 implies that in the worst case, these upper bounds are tight up to a constant factor.

4 The General Hidden Subgroup Problem

Finally, we consider the Hidden Subgroup Problem for a general finite group. Non-abelian groups are more varied in structure than abelian ones, and it appears challenging to extend the ideas used in the abelian case to them. Instead, we resort to a probabilistic argument to show the existence of a generating pair of small size. The construction comes within a poly-logarithmic multiplicative factor of optimal, and leads to our final algorithm.

Theorem 4.1. *For any finite group G with order n (with $n > 1$), there is a generating pair S_1, S_2 for G such that $|S_1|$ and $|S_2|$ are both at most $\lceil \sqrt{n \ln n} \rceil$.*

Proof. We let $S_1 := \{g_1, g_2, \ldots, g_t\}$, any subset consisting of t distinct group elements, where $t := \lceil \sqrt{n \ln n} \rceil$. Let $\boldsymbol{R} := \{\boldsymbol{h}_1, \boldsymbol{h}_2, \ldots, \boldsymbol{h}_t\}$ be a set of t distinct group elements chosen uniformly at random from the collection of t-element subsets of G. Recall that the set of products of elements from S_1 and $\boldsymbol{R}$ is denoted as $S_1\boldsymbol{R}$, i.e., $S_1\boldsymbol{R} := \{xy : x \in S_1,\ y \in \boldsymbol{R}\}$.

We show that for any fixed element $g \in G$, the probability that $g \notin S_1\boldsymbol{R}$ is less than $1/n$.

$$\begin{aligned}
\Pr(g \notin S_1\boldsymbol{R}) &= \Pr(\forall i \in [t],\ g_i^{-1}g \notin \boldsymbol{R}) \\
&= \binom{n-t}{t}\binom{n}{t}^{-1} \\
&= \frac{(n-t)(n-t-1)(n-t-2)\cdots(n-2t+1)}{n(n-1)(n-2)\cdots(n-t+1)} \\
&= \left(1-\frac{t}{n}\right)\left(1-\frac{t}{n-1}\right)\left(1-\frac{t}{n-2}\right)\cdots\left(1-\frac{t}{n-t+1}\right) \ .
\end{aligned}$$

Since $t > 1$, we get

$$\Pr(g \notin S_1\boldsymbol{R}) \quad < \quad \left(1-\frac{t}{n}\right)^t \quad < \quad \exp\left(-\frac{t^2}{n}\right) \quad \leq \quad \frac{1}{n} \ .$$

By the Union Bound, the probability that $S_1\boldsymbol{R} \neq G$ is strictly less than 1. So there is a subset S_2 of size t such that $S_1S_2 = G$. □

A similar result may be derived from a generalization of the Erdős-Rényi Theorem [7] due to Babai and Erdős [2]. However, this gives us a generating pair in which the size of a set may be as large as $4\sqrt{n \ln n}$.

As before, the algorithm for the general Hidden Subgroup Problem follows directly from the existence of a suitable generating pair for the underlying group.

Corollary 4.2. *There is a deterministic algorithm with query complexity at most $2\lceil\sqrt{n \ln n}\,\rceil$ that solves the Hidden Subgroup Problem over an arbitrary group G with order n. Moreover, if the input function hides the non-trivial subgroup H, the algorithm finds all the elements of H.*

Proof. By Theorem 4.1, there is a generating pair S_1, S_2 for the group G such that $|S_1|, |S_2| \leq \lceil\sqrt{n \ln n}\,\rceil$. We query the oracle function f at x^{-1} for all $x \in S_1$ and at all elements $y \in S_2$. If the function is injective on the set of points queried, we say f is injective. Otherwise, we say that f is not injective, and output $\{xy : x \in S_1,\ y \in S_2,\ f(x^{-1}) = f(y)\}$.

If the function is injective, the above algorithm outputs the correct answer. Suppose f hides the non-trivial subgroup H. Let $h \in H$ be any element of the hidden subgroup. We have $h = h_1h_2$ for some $h_1 \in S_1$ and $h_2 \in S_2$. By definition of H we have $f(h_1^{-1}) = f(h_2)$. When h is not the identity, we have $h_1^{-1} \neq h_2$ and the algorithm detects a collision. Moreover, the algorithm computes H correctly. □

We can obtain explicit, optimal algorithms for certain classes of non-abelian groups. For example, if a group G of order n has a subgroup H of order $\Theta(\sqrt{n}\,)$, then we can construct a generating pair S_1, S_2 of size $\Theta(\sqrt{n}\,)$ by taking $S_1 := H$ and S_2 to be a complete set of coset representatives of H. In fact, in this case, the group satisfies a stronger property; see, for example, Ref. [3]. Not all groups have a subgroup of such size. For example, the abelian group $\mathbb{Z}_p$ for a prime p does not have such a subgroup, and yet admits a suitable generating pair.

5 Subgroup-dependent query complexity

There is a randomized algorithm for the Hidden Subgroup Problem that has expected query complexity $O(\sqrt{|G| \,/\, |H|}\,)$ when the oracle f hides H, and has worst-case query complexity $O(\sqrt{|G|}\,)$.

In this section we use the notion of a generating pair in a more sophisticated manner to match this performance with a deterministic algorithm for all abelian groups, and some classes of non-abelian groups.

The algorithm rests on the following observation.

Lemma 5.1. *Suppose G_1 and H are subgroups of the possibly non-abelian finite group G such that G_1 has cardinality at least $|G| \,/\, |H|$. Then either $G_1 H = G$ or $|G_1 \cap H| > 1$.*

Proof. The intersection $G_1 \cap H$ is a subgroup of G and contains the identity element. There are exactly $|G| \,/\, |H|$ distinct left cosets of H. If $G_1 H \neq G$, by the Pigeon-Hole Principle, there are two distinct elements $g_1, g_2 \in G_1$ such that $g_1 H = g_2 H$, i.e., $g_2^{-1} g_1 \in H$. Since $g_2^{-1} g_1$ is also an element of G_1, and is not the identity element, we have $|G_1 \cap H| > 1$. □

Suppose that G is abelian, we know the order m of the hidden subgroup H, and $m > 1$. Then we may find a non-trivial element of H as follows. Consider any subgroup G_1 of G of order n/m, where $n := |G|$. Such a subgroup exists since G has a subgroup with order d for any positive divisor d of n [8, Corollary 2.4, page 77]. Let S_1, S_2 be a generating pair for G_1, and let g be any element of $G \setminus G_1$. By Lemma 5.1, either (i) there is a non-identity element $h \in H$ that is also in G_1, or (ii) $G_1 H = G$. In case (i), let $h = a + b$, where $a \in S_1$ and $b \in S_2$. If we query the oracle f at the elements $-x$ and y, for all $x \in S_1$ and $y \in S_2$, we will find $f(-a) = f(b)$, and can compute h. In case (ii), we have $g = g_1 + h$ for some $g_1 \in G_1$ and $h \in H$. Since g is not in G_1, the element h is not the identity. Suppose $g_1 = a + b$, with $a \in S_1$ and $b \in S_2$. Then $g - a = b + h$, and $f(g - a) = f(b)$. If we query the oracle f at the elements $g - x$ and y, for all $x \in S_1$ and $y \in S_2$, we will find $f(g - a) = f(b)$, and can compute h. This is the key idea underlying the algorithm.

For sets $S_1, S_2 \subseteq G$, define S_1^{-1} as the set $S_1^{-1} := \{x^{-1} : x \in S_1\}$. Following our notation for the product of sets of group elements, $S_1^{-1} S_2 = \{x^{-1} y : x \in S_1,\ y \in S_2\}$. Algorithm 1 (Find-Collision) implements the above idea with a geometrically decreasing sequence of guesses for the order of H. We show in Theorem 5.2 that the algorithm is correct and has the query complexity we seek for a large class of groups.

Theorem 5.2. *Algorithm 1 (Find-Collision) solves the Hidden Subgroup Problem over any finite group G. If the order of the group is n and that of the hidden subgroup H is m, the algorithm has query complexity as stated below.*

1. *If G is abelian, the algorithm has query complexity $\mathrm{O}(\sqrt{n/m}\,)$.*
2. *If G is not abelian, the algorithm has query complexity $\mathrm{O}(\sqrt{n \ln n}\,)$. Further, if G has a subgroup of order n_1 such that $n/m \leq n_1 \leq \kappa n/m$ for some $\kappa \geq 1$, then the query complexity is $\mathrm{O}(\sqrt{(\kappa n/m) \ln(\kappa n/m)}\,)$.*

Proof. Since the algorithm outputs a collision only when it finds one, it gives the correct answer when the oracle function f is injective. Suppose the function f hides a non-trivial subgroup H, so that $m \geq 2$.

If the group is abelian, it has a subgroup of order n/m [8, Corollary 2.4, page 77], and $n/m \leq n/2$. If it does not find a collision in an earlier iteration, Algorithm 1 finds a collision in an iteration with $k = \ell \geq 0$, where ℓ is such that $n/m \in \left[n/2^{\ell+1},\ n/(2^\ell + 1)\right]$. This is due to the reasoning given after Lemma 5.1. It thus outputs the correct answer. We have $\ell = \lceil \log_2 m \rceil - 1$, and initially, $k = \lceil \log_2 n \rceil - 1 \geq 0$. By Theorem 3.5, we have $|S_1|\,, |S_2| \leq 2\sqrt{|G_1|}$ in every iteration with queries.

Algorithm 1: Find-Collision(G, f)

Input : group G of order n, with $n > 1$; oracle for $f : G \to S$ that hides a subgroup
Output : injective, or collision $a, b \in G$

Let k be the integer l such that $n \in (2^l, 2^{l+1}]$;
if *G is abelian* **then** $k_0 \leftarrow 0$;
else $k_0 \leftarrow -1$;
while $k \geq k_0$ **do**
  Find, if there is one, a subgroup $G_1 \leq G$ with the largest order in $\left[n/2^{k+1},\ n/(\lfloor 2^k \rfloor + 1)\right]$; `/* the expression` $\lfloor 2^k \rfloor$ `is required to correctly handle the case` $k = k_0 = -1$ `*/`
  if *such a subgroup G_1 exists* **then**
    Find a generating pair S_1, S_2 for G_1 which minimizes $\max\{|S_1|,\ |S_2|\}$;
    if $G_1 = G$ **then** $g \leftarrow e$, the identity element of G ;
    else $g \leftarrow$ any element in $G \setminus G_1$;
    Let $R \leftarrow S_1^{-1} \cup S_2 \cup \left(S_1^{-1}\{g\}\right)$;
    Query f at all the elements in R ;
    if *$f(z) = f(y)$ for some $z, y \in R$ such that $z \neq y$* **then return** collision z, y ;
  $k \leftarrow k - 1$;
return injective

So the query complexity of the algorithm is at most

$$\begin{aligned}\sum_{k=\ell}^{\lceil \log_2 n \rceil - 1} 3 \cdot 2\sqrt{n/2^k} &\leq 6\sqrt{\frac{n}{2^\ell}} \sum_{i \geq 0} \frac{1}{\sqrt{2^i}} \\ &\leq 12(1+\sqrt{2})\sqrt{n/m}\ ,\end{aligned}$$

as $\ell \geq \log_2 m - 1$. The bound on the query complexity when $m = 1$ is the same as that for $m = 2$, as the algorithm executes all the iterations until $k = 0$. Part 1 of the theorem thus follows.

Suppose G is non-abelian. If the algorithm does not find a collision in earlier iterations, when $k = -1$, we have $G_1 = G$, and correctness follows as in Corollary 4.2. If G has a proper subgroup of order n_1 with $n/m \leq n_1 \leq \kappa n/m$ for some κ, the algorithm finds a collision as in the abelian case in an iteration with $k \geq \ell \geq 0$, where ℓ is such that $\kappa n/m \in \left[n/2^{\ell+1},\ n/(2^\ell + 1)\right]$. It thus outputs the correct answer. Further, we have $\ell = \lceil \log_2(m/\kappa) \rceil - 1$, and initially, $k = \lceil \log_2 n \rceil - 1 \geq 0$. By Theorem 4.1, we have $|S_1|, |S_2| \leq \sqrt{|G_1| \ln |G_1|} + 1$ in every iteration with queries. So the query complexity of the algorithm is at most

$$\begin{aligned}&\sum_{k=\ell}^{\lceil \log_2 n \rceil - 1} 3\left(1 + \left(\frac{n}{2^k} \ln \frac{n}{2^k}\right)^{1/2}\right) \\ &\leq 3(\lceil \log_2 n \rceil - \lceil \log_2(m/\kappa) \rceil + 1) + 3\left(\frac{n}{2^\ell} \ln \frac{n}{2^\ell}\right)^{1/2} \sum_{i \geq 0} \frac{1}{\sqrt{2^i}} \\ &\leq 6 + 3\log_2(\kappa n/m) + 3(2+\sqrt{2})\left(\frac{2\kappa n}{m} \ln \frac{2\kappa n}{m}\right)^{1/2}\ ,\end{aligned}$$

as $\ell \geq \log_2(m/\kappa) - 1$. The bound on the query complexity when $m = 1$ is $2\lceil \sqrt{n \ln n}\, \rceil$ more than

that for $m = 2$, as the algorithm executes all the iterations until $k = -1$. Part 2 of the theorem thus follows. □

Unlike abelian groups, a non-abelian group of order n may not have a subgroup of order n/m when it has a proper subgroup of order m. For example A_4, the alternating group of degree 4, has order 12, has several subgroups of order 2, but does not have a subgroup of order 6. However, in large classes of instances of the Hidden Subgroup Problem, subgroups of suitable size exist. An immediate example is the class of CLT groups. (A group G is called a *converse Lagrange Theorem (CLT) group* if it contains a subgroup of order d for every positive divisor d of $|G|$.) CLT groups include supersolvable groups; see, e.g., Ref. [9]. For such instances, Find-Collision achieves query complexity $O(\sqrt{(n/m)\ln(n/m)})$.

6 Finding the hidden subgroup

Unlike the algorithms in Corollary 3.6 and Corollary 4.2, Algorithm 1 may find only one non-trivial element from the hidden subgroup. In this section, we show how to extend Algorithm 1 to find the entire hidden subgroup.

When the group G is abelian, the problem of finding the entire subgroup may be reduced to that of finding one non-trivial element of the subgroup. This allows us to identify the subgroup by repeatedly using Algorithm 1 to find a set of generators.

Algorithm 2: Find-Abelian-Subgroup(G, f)

Input : group G of order n, with $n > 1$; oracle for $f : G \to S$ that hides a subgroup
Output : injective, or generators $S \subset G$ of the hidden subgroup

```
S ← ∅ ;
repeat
    H_1 ← ⟨S⟩, the subgroup generated by S ;
    G_1 ← G/H_1 ;
    if |G_1| > 1 then
        Let f_1 be the function defined by f and H_1 in the proof of Theorem 6.1 ;
        outcome ← Find-Collision(G_1, f_1) ;
        if outcome = collision a, b then S ← S ∪ {g_1^{-1} g_2}, where a = g_1 H_1 and b = g_2 H_1 ;
    else
        outcome ← injective
until outcome = injective;
if S = ∅ then return injective ;
else return generators S ;
```

Theorem 6.1. *There is a deterministic algorithm that solves the Hidden Subgroup Problem over any finite abelian group G, and finds the hidden subgroup with $O((\log m)\sqrt{n/m})$ queries when the order of G is n and that of the hidden subgroup is m.*

Proof. Suppose the oracle is f, the hidden subgroup is H, and we know a set of generators for a subgroup $H_1 \leq H$.

Define a function f_1 on the quotient group G/H_1 as $f_1(gH_1) := f(g)$ for any $g \in G$. The function f_1 is well-defined as left cosets of H_1 in G are subsets of left cosets of H in G, and the

function f is constant on left cosets of H. Moreover, the function f_1 hides the subgroup H/H_1 of G/H_1, as the left cosets of H in G correspond to left cosets of H/H_1 in G/H_1. Finally, the function f_1 may be evaluated with one query to the oracle for f.

Using this reduction, we may find the hidden subgroup H using Algorithm 2. The correctness of the algorithm follows by observing that in any iteration, if f_1 is injective, then H_1 equals the hidden subgroup. If f_1 is not injective, by Theorem 5.2, Find-Collision(G_1, f_1) returns a collision $g_1H_1, g_2H_1 \in G/H_1$ for f_1. Note that g_1, g_2 is a collision for f. We also have $g_1^{-1}g_2 \notin H_1$, so along with $g_1^{-1}g_2$, the set S generates a larger subgroup of H. Thus, the size of the subgroup H_1 increases by a factor of at least 2 in every iteration a collision is found, and the algorithm terminates after at most $\log_2 m$ iterations. In every iteration of Algorithm 2, the ratio of the order of G_1 and the hidden subgroup H/H_1 equals n/m. The query complexity of the algorithm now follows from Theorem 5.2. □

It is not clear how to extend Algorithm 2 to the non-abelian case, since the group H_1 may not be normal in general, and the corresponding quotient group may not be defined. We present a different algorithm, Algorithm 4, that works for some abelian *and* some non-abelian instances (which we describe after Theorem 6.3). Algorithm 4 builds on Algorithm 3, which is a variant of Algorithm 1 and is also based on Lemma 5.1.

Algorithm 3: Find-New-Collision(G, H_1, f)

Input : group G of order n, with $n > 1$; subgroup $H_1 \leq G$; oracle for $f : G \to S$ that hides a subgroup containing H_1

Output : no-new-collision, or collision $a, b \in G$ such that $a^{-1}b \notin H_1$

Let k be the integer l such that $n \in (2^l, 2^{l+1}]$;
if *G is abelian* **then** $m_1 \leftarrow \max\{2, |H_1|\}$;
else $m_1 \leftarrow |H_1|$;
$k_0 \leftarrow \lceil \log_2 m_1 \rceil - 1$;
while $k \geq k_0$ **do**
    Find, if there is one, a subgroup $G_1 \leq G$ with the largest order in $\left[n/2^{k+1},\ n/(\lfloor 2^k \rfloor + 1)\right]$ such that $G_1 \cap H_1 = \{e\}$, where e is the identity element of G ; `/* the expression` $\lfloor 2^k \rfloor$ `is required to correctly handle the case` $k = k_0 = -1$ `*/`
    if *such a subgroup G_1 exists* **then**
        Find a generating pair S_1, S_2 for G_1 which minimizes $\max\{|S_1|,\ |S_2|\}$;
        if $G_1H_1 = G$ **then** $g \leftarrow e$, the identity element of G ;
        else $g \leftarrow$ any element in $G \setminus (G_1H_1)$;
        Let $R \leftarrow S_1^{-1} \cup S_2 \cup \left(S_1^{-1}\{g\}\right)$;
        Query f at all the elements in R ;
        if *$f(z) = f(y)$ for some $z, y \in R$ such that $z^{-1}y \notin H_1$* **then return** collision z, y ;
    $k \leftarrow k - 1$;
return no-new-collision

Theorem 6.2. *Let (G, f) be an instance of the Hidden Subgroup Problem, H the subgroup that f hides, and H_1 a subgroup of H. Let the orders of G, H, H_1 be n, m, m_1, respectively. Suppose G has a subgroup G_0 of order n_0 such that $n_0 \geq n/m$, and G_0 intersects H_1 only in the identity element. Then Find-New-Collision(G, H_1, f) (Algorithm 3) returns "no new collision" if $H_1 = H$, and returns a collision $a, b \in G$*

such that $a^{-1}b \notin H_1$ otherwise. Further, if $n_0 \leq \kappa n/m$ for some $\kappa \geq 1$, the algorithm has query complexity as stated below:

1. *if G is abelian, the algorithm has query complexity $O(\sqrt{\kappa n/m})$, and*
2. *if G is not abelian, the algorithm has query complexity $O(\sqrt{(\kappa n/m)\ln(\kappa n/m)})$.*

Proof. When $H_1 = \{e\}$, the algorithm is identical to Algorithm 1, and its correctness follows by Theorem 5.2. Suppose H_1 is not the trivial subgroup.

Since the algorithm reports a collision a, b only when $a^{-1}b \notin H_1$, it gives the correct answer when $H_1 = H$. Suppose $H_1 \neq H$. By hypothesis, G has a subgroup G_0 of order at least n/m such that G_0 intersects H_1 only in the identity element. Then $|G_0| \leq n/m_1 \leq n/2$; otherwise, we would have two distinct elements of G_0 in the same coset of H_1, which implies that $|G_0 \cap H_1| > 1$.

If it does not find a collision in earlier iterations, the algorithm finds a subgroup G_1 with $|G_1| \geq n/m$ and $G_1 \cap H_1 = \{e\}$ in an iteration with $k \geq k_0 \geq 0$ (recall that $k_0 := \lceil \log_2 m_1 \rceil - 1 \geq 0$, since $m_1 \geq 2$). Let S_1, S_2 be the generating pair for G_1 computed by the algorithm.

Suppose $G_1H_1 = G$. Consider any element $h \in H \setminus H_1$. We have $h = g_1h_1$ for some $g_1 \in G_1$ and $h_1 \in H_1$. So $g_1 = hh_1^{-1} \in G_1 \cap H$, and $g_1 \notin H_1$. We also have $g_1 = ab$ for some $a \in S_1$ and $b \in S_2$, so $f(a^{-1}) = f(b)$. The algorithm queries the oracle f at the elements x^{-1} and y, for all $x \in S_1$ and $y \in S_2$, so it finds and returns a collision.

Now suppose $G_1H_1 \neq G$, so that the algorithm also finds an element $g \in G \setminus (G_1H_1)$ in the same iteration.

If $G_1H = G$, we have $g = g_1h$ for some $g \in G_1$ and $h \in H$. Since $g \notin G_1H_1$, $g_1h \notin G_1H_1$, and $h \notin H_1$. Suppose $g_1 = ab$, with $a \in S_1$ and $b \in S_2$. Then $a^{-1}g = bh$, and $f(a^{-1}g) = f(b)$. The algorithm queries the oracle f at the elements $x^{-1}g$ and y, for all $x \in S_1$ and $y \in S_2$, so it finds $f(a^{-1}g) = f(b)$, and returns a collision.

If $G_1H \neq G$, as in Lemma 5.1, there are two distinct elements $g_1, g_2 \in G_1$ such that $g_2^{-1}g_1 \in H$. Since $G_1 \cap H_1 = \{e\}$, and $g_1 \neq g_2$, we have $g_2^{-1}g_1 \in H \setminus H_1$. We also have $g_2^{-1}g_1 = ab$, for some $a \in S_1$ and $b \in S_2$. The algorithm queries the oracle f at the elements x^{-1} and y, for all $x \in S_1$ and $y \in S_2$, so it finds $f(a^{-1}) = f(b)$, and returns a collision in this case as well.

Assuming $n/m \leq |G_0| \leq \kappa n/m \leq n$ for some $\kappa \geq 1$ with G_0 as in the statement of the theorem, the algorithm executes iterations with $k \geq \ell$, where ℓ is such that $\kappa n/m \in \left[n/2^{\ell+1},\ n/(\lfloor 2^\ell \rfloor + 1)\right]$. So the query complexity of the algorithm follows by same kind of analysis as in Theorem 5.2. □

Thus, whenever there is a large enough subgroup G_0 that intersects with a proper subgroup H_1 of the hidden subgroup H only in the identity element, Algorithm 3 gives us an element h of H that is not in H_1. Along with H_1, the element h generates a strictly larger subgroup of H. As long as the condition above holds for all proper subgroups of H, we can repeat Algorithm 3 until we find a set of generators for the hidden subgroup. This process is described in Algorithm 4.

Theorem 6.3. *Algorithm 4 (Find-Subgroup) solves the Hidden Subgroup Problem over a group G and finds the hidden subgroup H when G has a subgroup G_0 of order n/m such that $|G_0 \cap H| = 1$, where n and m are the orders of G and H, respectively. Moreover, the algorithm makes*

- $O((\log m)\sqrt{n/m})$ *queries when G is abelian, and*
- $O((\log m)\sqrt{(n/m)\log(n/m)})$ *queries when G is non-abelian.*

Proof. The existence of a subgroup G_0 as in the statement of the theorem implies that all the hypotheses of Theorem 6.2 are satisfied for every subgroup H_1 of H. So, starting with $H_1 = \{e\}$, where e is the identity element of G, Find-New-Collision(G, H_1, f) returns a collision a, b in each

Algorithm 4: Find-Subgroup(G, f)

Input : group G of order n, with $n > 1$; oracle for $f : G \to S$ that hides a subgroup
Output : injective, or generators $S \subset G$ of the hidden subgroup

$S \leftarrow \emptyset$;
repeat
  $H_1 \leftarrow \langle S \rangle$;
  outcome $\leftarrow$ Find-New-Collision(G, H_1, f) ;
  if *outcome* = collision a, b **then** $S \leftarrow S \cup \{a^{-1}b\}$;
until *outcome* = injective;
if $S = \emptyset$ **then return** injective ;
else return generators S ;

iteration of Algorithm 4 in which $H_1 \neq H$. Since $a^{-1}b \notin H_1$, the set $S \cup \{a^{-1}b\}$ generates a larger subgroup of H. Thus, the size of the subgroup H_1 increases by a factor of at least 2 in every iteration in which a collision is found, and the algorithm terminates after at most $\log_2 m$ iterations. The query complexity follows from Theorem 6.2. □

A subgroup as in the statement of Theorem 6.3 exists if G is the semidirect product of H with another subgroup, i.e., H is normal and there is a subgroup K such that $G = H \rtimes K$, or there is a normal subgroup K such that $G = H \ltimes K$. We may then take $G_0 := K$. Not all groups have a semidirect product structure, even if the hidden subgroup is normal. For example, every proper subgroup H of the abelian group $\mathbb{Z}_{p^k}$ with $k > 1$ is normal, but $\mathbb{Z}_{p^k}$ cannot be expressed as a semidirect product of H with another subgroup. On the other hand, a group need not have a semidirect product structure for a subgroup with the properties in Theorem 6.3 to exist. For example, consider S_n, the symmetric group of degree n, and S_{n-1} as its subgroup consisting of permutations that map n to itself. Then $S_n = S_{n-1}H$, where H is the subgroup generated by the cycle $(1\ 2\ 3\ \cdots\ n)$, and neither S_{n-1} nor H is normal in S_n for $n \geq 4$. Such groups are known as the *bicrossed* products (also as *Zappa-Szép* or *knit* products); see, e.g., [1, 4]. Thus, Algorithm 4 finds the hidden subgroup H with query complexity as in Theorem 6.3 whenever G is the bicrossed product of H with another group. A description of groups arising as bicrossed products is a matter of ongoing research [1].

7 Open problems

We conclude with a few open problems. The query complexity of the algorithm designed by Ye and Li [14] for finding the hidden subgroup in abelian instances may be smaller than that of Algorithm 2. The lower query complexity hinges on an intricate analysis of the structure of the hidden subgroup. Can we establish the same query complexity through simpler means?

There are a number of variants of the Hidden Subgroup Problem, for example, when the underlying group is specified implicitly. These may also admit deterministic algorithms with optimal classical query complexity. The precise characterization of the deterministic query complexity of the Hidden Subgroup Problem for explicitly specified groups, especially in the non-abelian case, is perhaps the most interesting problem left open by this work. Related questions are whether there is a generating pair of size $O(\sqrt{n})$ for any non-abelian group of order n, for what instances of the problem Algorithms 1 and 3 give the correct output with query complexity $\widetilde{O}(\sqrt{n/m})$, where m is

the order of the hidden subgroup, or whether there are similar "generic" algorithms that achieve this query complexity for larger classes of groups.

References

[1] A. L. Agore, A. Chirvăsitu, B. Ion, and G. Militaru. Bicrossed products for finite groups. *Algebras and Representation Theory*, 12(2):481–488, October 1, 2009.

[2] László Babai and Paul Erdős. Representation of group elements as short products. In Peter L. Hammer, Alexander Rosa, Gert Sabidussi, and Jean Turgeon, editors, *Theory and Practice of Combinatorics*, volume 60 of *North-Holland Mathematics Studies*, pages 27–30. North-Holland, 1982.

[3] Kady Hossner Boden and Michael B. Ward. The heritage of Cayley-Sudoku tables. Technical Report arXiv:2001.06711v1 [math.GR], arXiv.org, January 2020.

[4] Matthew G. Brin. On the Zappa-Szép product. *Communications in Algebra*, 33(2):393–424, 2005.

[5] Guangya Cai and Daowen Qiu. Optimal separation in exact query complexities for Simon's problem. *Journal of Computer and System Sciences*, 97:83–93, 2018.

[6] Andrew M. Childs and Wim van Dam. Quantum algorithms for algebraic problems. *Reviews of Modern Physics*, 82:1–52, January 2010.

[7] Paul Erdős and Alfréd Rényi. Probabilistic methods in group theory. *Journal d'Analyse Mathématique*, 14(1):127–138, December 1, 1965.

[8] Thomas W. Hungerford. *Algebra*, volume 73 of *Graduate Texts in Mathematics*. Springer-Verlag, New York, 1974.

[9] D. H. McLain. The existence of subgroups of given order in finite groups. *Mathematical Proceedings of the Cambridge Philosophical Society*, 53(2):278–285, 1957.

[10] Ashwin Nayak. Deterministic algorithms for the Hidden Subgroup Problem. Technical Report arXiv:2104.14436v1 [cs.DS], arXiv.org, April 2021.

[11] Daniel R. Simon. On the power of quantum computation. *SIAM Journal on Computing*, 26(5):1474–1483, 1997.

[12] Zhenggang Wu, Daowen Qiu, Jiawei Tan, Hao Li, and Guangya Cai. Quantum and classical query complexities for Generalized Simon's Problem. Technical Report arXiv:1905.08549v2 [quant-ph], arXiv.org, September 2021.

[13] Zekun Ye, Yunqi Huang, Lvzhou Li, and Yuyi Wang. Query complexity of Generalized Simon's Problem. *Information and Computation*, page 104790, 2021.

[14] Zekun Ye and Lvzhou Li. Deterministic algorithms for the hidden subgroup problem. Technical Report arXiv:2110.00827v1 [cs.DS], arXiv.org, October 2021.