Tabulating pseudoprimes and tabulating liars

Andrew Shallue

Tabulating Pseudoprimes and Tabulating Liars

Andrew Shallue

1 Abstract

This paper explores the asymptotic complexity of two problems related to the Miller-Rabin-Selfridge primality test. The first problem is to tabulate strong pseudoprimes to a single fixed base a. It is now proven that tabulating up to x requires $O(x)$ arithmetic operations and $O(x \log x)$ bits of space. The second problem is to find all strong liars and witnesses, given a fixed odd composite n. This appears to be unstudied, and a randomized algorithm is presented that requires an expected $O((\log n)^2 + |S(n)|)$ operations (here $S(n)$ is the set of strong liars). Although interesting in their own right, a notable application is the search for sets of composites with no reliable witness.

2 Introduction

A common way to prove that a positive integer n is composite is to demonstrate that a necessary condition for primality fails. For example, by Fermat's Theorem we know that if p is prime and a is not divisible by p, then $a^{p-1} \equiv 1 \mod p$. The converse of Fermat's Theorem fails, inspiring the following definition.

Definition 2.1. Let n be a positive, composite integer. If $a^{n-1} \equiv 1 \mod n$ we call a a Fermat liar with respect to n, and we call n a Fermat a-pseudoprime or a-psp for short. If $a \not\equiv 0 \mod n$ and $a^{n-1} \not\equiv 1 \mod n$ then a is a Fermat witness for n.

If $a \equiv 0 \mod n$ then a is neither a witness nor a liar.

Due to the existence of composite numbers with many Fermat liars, a stronger condition was developed by Selfridge, Miller, Rabin and others. Let $n > 1$ be odd, and write $n-1$ as $2^s d$ where d is odd. Then if n is prime and $\gcd(a, n) = 1$ we have that either

1. $a^d \equiv 1 \mod n$, or
2. $a^{2^i d} \equiv -1 \mod n$ for some $0 \le i \le s-1$.

We call this the strong pseudoprime condition. An algorithm built around this condition is known as the Miller-Rabin-Selfridge test, and it inspires a similar definition of liar.

Definition 2.2. Let n be an odd, positive, composite integer. If a satisfies the strong pseudoprime condition, we call a a strong liar with respect to n, and we call n a strong a-pseudoprime or a-spsp for short. If $a \not\equiv 0 \mod n$ and a does not satisfy the condition then a is a strong witness for n.

If $a \equiv 0 \mod n$ then a is neither a strong witness nor a strong liar.

Note that 1 and $n-1$ are strong liars for any odd composite n. Excluding the case when $\gcd(a,n)=n$, if $\gcd(a,n) \neq 1$ then a must be a strong witness. Since $1, -1$ are units modulo n and a is not, no power of a could result in 1 or -1 modulo n.

If n fixed we denote the set of Fermat liars by $F(n)$ and the set of strong liars by $S(n)$. From [20] we have exact formulae for the size of these sets, namely

$$|F(n)| = \prod_{p|n} \gcd(n-1, p-1) \qquad\qquad |S(n)| = \left(1 + \frac{2^{\nu\omega(n)}-1}{2^{\omega(n)}-1}\right) \prod_{p|n} \gcd(n', p')$$

where $\omega(n)$ is the number of distinct primes dividing n, $\nu = \min_{p|n} \nu_2(p-1)$, and n' is the odd part of $n-1$.

Some authors in defining a liar a insist upon $a < n$, a reflection of the obstruction they pose to identifying composite numbers. In this paper we seek to classify the set of all positive integers, and will allow $a > n$. It is natural in this case to work with a modulo n, and the definition given requires no modification to allow for it.

This paper will focus on the asymptotic complexity of two tabulation problems related to strong liars. First, given a fixed a, tabulate all a-spsp up to a given bound x. Second, given a fixed odd composite n, determine all $1 \leq a < n$ for which a is a strong liar. While many authors have performed pseudoprime searches or pseudoprime tabulations (especially pseudoprimes to multiple bases), fewer analyze algorithms for these problems in terms of their asymptotic complexity. In fact, the literature seems to lack algorithms for tabulating strong witnesses and liars. The naive complexity to beat is that of simply applying the strong pseudoprime condition, either to each odd composite up to x or to each potential liar a. This method requires the equivalent of a single modular exponentiation for each integer, which totals to $O(x \log x)$ multiplications.

There has been interest in tabulating pseudoprimes since a variety of primality tests were first developed. A significant piece of prior work is [24], which contains a tabulation of 2-Fermat pseudoprimes up to $25 \cdot 10^9$. The key algorithmic idea was to restrict the search to certain equivalence classes. In particular, if n is a 2-psp with $p \mid n$ then $\ell_2(p) \mid n-1$ (here $\ell_2(p)$ is the multiplicative order of 2 modulo p). This tabulation was extended to 10^{13} in [22]. Pinch added a pre-computation that matched integers f with all primes q such that $\ell_a(q) = f$, then looped over pre-products P, factors f of $P-1$, and primes q with $\ell_a(q) = f$ in search of pseudoprimes Pq. A method similar to Pinch's has been used to tabulate all 2-psp up to $2^{64}-1$ [12]. One contribution of the present work is demonstrating that $\ell_a(q)$ can be computed for all prime powers $q \leq x$ at a cost of $O(x)$ multiplications.

More recent tabulations have focused on the search for ψ_k, defined as the smallest composite which is a strong pseudoprime to all of the first k prime bases. The resulting specialized algorithms take advantage of the fact that pseudoprimes to several bases fall into an even more restricted set of congruence classes. Jaeschke in [17] utilized this technique to tabulate $n \leq 10^{12}$ which are simultaneously 2-spsp, 3-spsp, and 5-spsp. Zhang [31] further restricted the search to pseudoprimes with two prime factors related by a single parameter, finding all such pseudoprimes to 10^{36}. In the process, upper bounds or exact values were given for ψ_k with $k \leq 19$. The authors of [18] tabulated pseudoprimes to the first eight prime bases and provably determined ψ_{11}. An algorithm developed in [6] tabulates pseudoprimes to two or more bases up to x and has a heuristic complexity of $O(x^{9/11+o(1)})$. Recently, Sorenson and Webster improved this to $O(x^{2/3+o(1)})$ and computed ψ_{12} and ψ_{13} [28].

Two main results will be discussed in later sections. First, an algorithm is presented in Section 6 that returns all a-spsp up to x. For the first time it is proven that tabulating strong pseudoprimes to a single fixed base a requires $O(x)$ arithmetic operations and $O(x \log x)$ bits of space. The second new result is an algorithm which tabulates all strong liars for a given odd composite n in factored form. Presented as Theorem 8.2 in Section 8, we first use a randomized algorithm to compute the structure of $F(n)$ at an expected cost of $O((\log n)^2)$ operations, after which tabulating $S(n)$ requires $O(|S(n)|)$ operations.

3 Notation and model of computation

For a in the unit group $(\mathbb{Z}/n\mathbb{Z})^\times$, $\ell_a(n)$ will denote the multiplicative order of a modulo n. That is, $\ell_a(n)$ is the least integer e such that $a^e \equiv 1 \mod n$. By Lagrange's Theorem $\ell_a(n) \mid \phi(n)$, where $\phi(n)$ is Euler's function.

The largest power of p that divides n will be notated by $\nu_p(n)$. Note that one way to compute $\ell_a(n)$ is to compute $\nu_p(\ell_a(n))$ for all primes p dividing $\phi(n)$. The number of distinct prime factors of an integer n is denoted $\omega(n)$ and the largest prime factor of n by $P(n)$.

In this paper log will stand for the natural logarithm, though $\log_2$ will occasionally be used in describing algorithms.

The computational model used assumes all arithmetic operations take constant time, though the space complexity of algorithms will always be measured in bits. In particular, if x is the bound on the tabulation being performed, all integers considered have $O(\log x)$ bit size, and we assume arithmetic operations on such integers take constant time. See [10, Section 2] for further details and justification.

4 Reliable witnesses

The inspiration for the present work comes from a question posed in [1]: find the smallest integer x for which there is no reliable strong witness for the set of odd composites up to x. This seems difficult, but if we change the problem to "no reliable witness less than or equal to x" it becomes more tractable.

Definition 4.1. Let $\mathcal{C}$ be a set of composite integers. We call positive integer a a reliable Fermat witness for $\mathcal{C}$ if it is a Fermat witness for every composite $n \in \mathcal{C}$. We similarly define reliable strong witness, and use simply "reliable witness" if the test is clear from context.

A little computation reveals that the answer to the modified question is 2047. For the first 2-spsp is 2047, so 2 is a reliable witness for the odd composites less than 2047. Looking at $a < 2047$, the smallest a-spsp is always smaller than 2047, except for the fact that the smallest 1320-spsp is 4097. However, 1320 is not a reliable witness, because it has odd composite divisors. For such an odd composite divisor n, we have $1320 \equiv 0 \mod n$ and thus 1320 is neither a witness nor a liar. We conclude that the odd composites up to 2047 have no reliable witness $\leq x$, and that this is the smallest bound with that property.

A natural extension is to construct a small set of odd composites $\leq x$ with no reliable witness $\leq x$ for arbitrary x. By work of Monier [20] and others, we know that the maximum number of strong liars a composite n can possess is $\phi(n)/4$. By using n that reach the maximum we should have a set of composites with few reliable witnesses. For each such reliable witness a, add the smallest a-spsp until the set no longer has a reliable witness. Note that implementing this construction requires algorithms for the two problems under consideration, namely tabulating all a-spsp up to a given bound x and determining all $1 \leq a < n$ for which a is a strong liar.

There are known theoretical bounds on the count of a-pseudoprimes up to x, denoted by $\mathcal{P}_a(x)$. Since every Carmichael number relatively prime to a is an a-pseudoprime, [2] gives a lower bound of $x^{2/7}$. Combined with an upper bound from [23], we have

$$x^{2/7} \leq \mathcal{P}_a(x) \leq x/\sqrt{\exp(\log x \cdot \log\log\log x/\log\log x)}$$

for large enough x.

Useful theoretical results are also available on counts of Fermat liars and strong liars (here we restrict to liars a with $1 \leq a \leq n-1$). Carmichael numbers are composite integers n that are a-psp for all a relatively prime to n. Thus by [2] there are infinitely many composite n with $\phi(n)$ Fermat liars. The size of $\phi(n)$ depends on how many prime factors n possesses. For each fixed k, there are infinitely many Carmichael numbers n with no prime factor below $(\log n)^k$. For such n, the probability that $\gcd(a,n) \neq 1$ is at most $(\log n)^{-k+1}$, where $1 \leq a \leq n-1$ is chosen uniformly at random [1, Section 2]. See [13] for more results and conjectures on counts of Carmichael numbers with a specified number of prime factors. Bounds on the normal order and average order of the set of strong liars can be found in [11].

Related to reliable witnesses is the notion of the least strong witness for an odd composite n, denoted $W(n)$. Making explicit a result of Miller, Bach in [4] showed that $W(n) < 2(\log n)^2$ under the Extended Riemann Hypothesis. For a lower bound, $W(n) > (\log n)^{1/(3 \log\log\log n)}$ for infinitely many Carmichael numbers n [1]. More generally, the authors showed that for x sufficiently large and for any set of potential witnesses not too large, there are Carmichael numbers smaller than x with no witness in the set. Thus asymptotically there cannot be a reliable witness for the set of odd composites up to x.

5 Preliminaries

The algorithms developed in later sections will require access to the factorization of any integer in a given range. The goal is to create an array that stores in position i the smallest prime factor of i. Then, one can retrieve the full factorization of any n with $O(\log n)$ division steps through finding the prime factor p and recursively finding the smallest prime factor of n/p.

A factor table for integers up to x can be created with $O(x \log\log x)$ arithmetic operations using the Sieve of Eratosthenes [9, Section 3.2.3]. There is an extensive literature on linear and sublinear prime sieves one can turn to for an improvement [10, 25, 26]; the following theorem only adds the idea of a static wheel to the Sieve of Eratosthenes.

Lemma 5.1. *Given a bound x, there exists an algorithm that outputs an array A of size x that contains at position i the smallest prime factor of i. This algorithm requires $O(x)$ arithmetic operations and $O(x \log x)$ bits of space.*

Proof. For an introduction to the wheel datastructure, see [10, Section 2]. In our case, in addition to storing the distance to the next integer relatively prime to the first k primes, we need to compute and store the smallest prime factor for the integers up to the wheel size $m = \prod_{i=1}^{k} p_i$ which are divisible by one of the first k primes. If we pick k so that $\prod_{i=1}^{k} p_i$ is at most $\sqrt{x}$, this can be done using the Sieve of Eratosthenes in time $O(\sqrt{x} \log\log\log x)$ or even in time $O(\sqrt{x})$ [10, Lemma 1]. Let array W store the distance to the next relatively prime integer, and array R store the smallest prime factor for integers up to m.

Now, as a first step, for each $i < n$ set $A[i] = R[i \bmod m]$. As a second step, for each prime $p > p_k$, and for each f relatively prime to m, set $A[pf] = p$ if p is smaller than the previous entry of $A[pf]$.

This algorithm is correct, since if the smallest prime factor of i is $p \le p_k$, then p is also the smallest prime factor of $i \bmod m$. If the smallest prime factor of i is greater than p_k, it is caught in Step 2.

Step 1 requires $O(x)$ arithmetic operations. Step 2 uses the wheel to step through the integers relatively prime to m in linear time. The number of such integers is

$$x \cdot \frac{\phi(m)}{m} = x \prod_{i=1}^{k} \left(1 - \frac{1}{p_i}\right) = O\left(\frac{x}{\log\log x}\right)$$

[5, Theorem 8.8.6] and so the cost of Step 2 in operations is

$$O\left(\sum_{p_k < p < x} \left\lfloor \frac{x}{p \log\log x} \right\rfloor\right) \leq O\left(\frac{x}{\log\log x} \sum_{p<x} \frac{1}{p}\right) = O(x)$$

[5, Theorem 8.8.5]. □

In what follows, we will assume that such an algorithm has been run, and hence that we have access to any prime factorization.

To apply sieve techniques we need a characterization of strong pseudoprimes that depends upon properties of their prime factors. This first proposition shows that strong pseudoprimes are not much harder to identify than Fermat pseudoprimes.

Proposition 5.2 ([1]). *Let n be a positive, odd composite integer. Then n is a strong pseudoprime to the base a if and only if $a^{n-1} \equiv 1 \mod n$ and there exists an integer e such that, for every prime factor p of n, $\nu_2(\ell_a(p)) = e$.*

We now extend the idea in [24], characterizing a Fermat pseudoprime n by the prime powers that divide it.

Proposition 5.3. *Suppose the factorization of n is $\prod_{i=1}^{k} p_i^{r_i}$. Then $a^{n-1} \equiv 1 \mod n$ if and only if $n-1$ is a multiple of $\ell_a(p_i^{r_i})$ for all $1 \leq i \leq k$.*

Proof. Suppose that $a^{n-1} \equiv 1 \mod n$. Then since $p_i^{r_i} \mid n$, $a^{n-1} \equiv 1 \mod p_i^{r_i}$, and by group theory $\ell_a(p_i^{r_i}) \mid n-1$.

Conversely, if $n-1$ is a multiple of $\ell_a(p_i^{r_i})$, then $a^{n-1} \equiv 1 \mod p_i^{r_i}$, and since the factorization of n is $\prod_i p_i^{r_i}$, we use the Chinese Remainder Theorem to conclude that $a^{n-1} \equiv 1 \mod n$. □

This gives a restricted set of equivalence classes to which pseudoprimes belong.

Proposition 5.4. *Assume that $\gcd(p^r, \ell_a(p^r)) = 1$, so that $\ell_a(p^r) \mid p-1$. Then $p^r \mid n$ and $\ell_a(p^r) \mid n-1$ both hold if and only if $n \equiv p^r \mod p^r \cdot \ell_a(p^r)$.*

Proof. First assume that $n \equiv 0 \mod p^r$ and $n \equiv 1 \mod \ell_a(p^r)$. Then $\ell_a(p^r) \mid p-1$ implies $\ell_a(p^r) \mid p^r - 1$, and it follows that $\ell_a(p^r) \mid (n-1-(p^r-1))$. So $\ell_a(p^r)$ divides $n - p^r$. Since p^r also divides $n - p^r$ and $\gcd(p^r, \ell_a(p^r)) = 1$, the product divides $n - p^r$ and we conclude that $n \equiv p^r \mod p^r \cdot \ell_a(p^r)$.

Now suppose that $n \equiv p^r \mod p^r \cdot \ell_a(p^r)$. Then p^r divides $n - p^r$, and thus p^r divides n. We also have $\ell_a(p^r) \mid (n-1-(p^r-1))$ and so $\ell_a(p^r)$ divides $n-1$. □

We will need to compute $\ell_a(p^r)$ for all prime powers p^r up to x. With access to the factorization of $\phi(p^r)$, the preferred algorithm is a known method that determines the power of each prime dividing the order [19, Algorithm 4.79].

Algorithm 1: Computing multiplicative order

```
Input  : positive integers $n, a$, and factorization $\phi(n) = p_1^{r_1} \cdots p_w^{r_w}$
Output: $\ell_a(n)$
1 for $1 \le i \le w$ do
2     Compute $\alpha_i = a^{\phi(n)/p_i^{r_i}} \mod n$ ;
3     Compute $\alpha_i^{p_i^e} \mod n$ for $0 \le e \le r_i$ until the result is 1. Let $q_i$ be the first such
      power $p_i^e$ ;
4 return $\prod_{i=1}^{w} q_i$
```

As Sutherland notes in [29, Section 7.3], line 2 in Algorithm 1 requires w exponentiations while line 3 requires $O(\log n)$ multiplications over the entire loop. Since we are repeatedly powering the same base a in the same group, it makes sense to apply precomputation to reduce the number of multiplications required.

Lemma 5.5 ([7])**.** *Let g be an element of a group of order N. If $O(\log N/\log\log N)$ powers are precomputed, then we may compute g^n using only*

$$(1+o(1))\frac{\log_2 N}{\log_2\log_2 N}$$

group operations.

As a corollary, the asymptotic running time of Algorithm 1 is

$$O\left(\log n + \omega(n)\frac{\log n}{\log\log n}\right)$$

multiplications.

Next, it is necessary to know how many distinct prime factors $\phi(p^r) = p^{r-1}(p-1)$ has on average.

Lemma 5.6. *The following sum is over prime powers. We have*

$$\sum_{p^r \le x} \omega(\phi(p^r)) = O\left(\frac{x\log\log x}{\log x}\right) .$$

Proof. From [15] we know that

$$\sum_{p \le x} \omega(p-1) = O\left(\frac{x\log\log x}{\log x}\right) .$$

Note that since $\phi(p^r) = p^{r-1}(p-1)$ we have $\omega(\phi(p^r)) = \omega(p-1)+1$ and hence $\sum_{p \le x} \omega(\phi(p^r)) = O(x \log\log x / \log x)$.

Now consider prime powers p^r with $r > 1$. Then

$$\sum_{\substack{p^r \le x \\ r \ge 2}} \omega(p-1) + 1 = \sum_{r=2}^{\log_2 x} \sum_{p \le x^{1/r}} \omega(p-1) + 1 = O\left(\log x \cdot \frac{\sqrt{x} \log\log x}{\log x} + \frac{\sqrt{x}}{\log x} \log x\right)$$

which is asymptotically smaller. This completes the proof. □

6 Computing a-pseudoprimes

This section presents a new algorithm for finding all a-spsp up to a bound x. As a corollary, an easy modification gives an algorithm with the same running time that computes all a-psp up to x. The authors of [24] used one direction of Proposition 5.3 to reduce the number of pseudoprime tests required. Our improvement is to consider all prime powers, and the key observation is that computing $\ell_a(p^r)$ for all prime powers does not spoil the asymptotic complexity. Pinch in [22] considered a similar sieving strategy, but computed orders in a different way that is hard to analyze. Computing strong pseudoprimes rather than Fermat pseudoprimes does not add much difficulty; since Fermat pseudoprimes are rare we simply apply the strong pseudoprime test to each Fermat pseudoprime.

Note that on average the number of distinct prime factors of n is $O(\log\log n)$, and so a worst case analysis of the loop in line 12 would be $O(x \log\log x)$ operations. The success of this algorithm depends upon the fact that most composites are not pseudoprimes, and that non-pseudoprimes will be discovered rather rapidly, so that on average only a constant number of factors need to be checked before an obstruction to a composite integer's pseudoprimality is found. The author is grateful to Jonathan Sorenson for this insight.

Theorem 6.1. *Given base a and bound x, Algorithm 2 correctly computes all a-spsp up to x. The algorithm has a time complexity of $O(x)$ arithmetic operations and requires $O(x \log x)$ bits of space.*

Proof. In lines 4 and 5 the algorithm discards cases where n is prime, n is even, or a is not a unit modulo n. If n fails Proposition 5.3 and is thus not a Fermat pseudoprime it is caught at line 14, while at line 16 we catch composites which are Fermat pseudoprimes but not strong pseudoprimes. Thus if Algorithm 2 sets $P[n] = 0$, n is indeed not a strong pseudoprime.

Conversely, if $P[n] = 1$ at the end of the algorithm, then n is a strong pseudoprime to the base a. For if n has passed the line 12 while loop, then $\ell_a(p^r) \mid n-1$ for all prime powers p^r dividing n, making n an a-psp by Proposition 5.3. Since $P[n] = 1$, we know that n also passed the strong pseudoprime test at line 16.

Algorithm 2: Tabulating strong pseudoprimes

```
Input  : positive integer $a$, bound $x$
Output: boolean array $P[]$ where $P[n] = 1$ if $n$ is a spsp($a$)

Use Lemma 5.1 to create an array containing smallest prime factors ;
Initialize a boolean array $P[]$ of size $x$ with all entries as 1 ;
for primes $p \le x$ do
    Set $P[p] = 0$ ;                                    // primes are not pseudoprimes
    if $p \mid a$ or $p == 2$ then
        set $P[n] = 0$ for all multiples $n$ of $p$ ;     // require $\gcd(n, a) = 1$, $n$ odd
    for prime powers $q = p^r$ do
        Factor $\phi(q)$ using array from line 1 ;
        Compute $\ell_a(q)$ using Algorithm 1 ;
for $n = 2$ to $x$ with $P[n] == 1$ do
    Set $q = p^r$ to be the prime power dividing $n$ with smallest $p$ ;
    while $n \equiv 1 \mod \ell_a(q)$ do
        Set $q$ to be the prime power dividing $n$ with the next smallest $p$ ;
    if $n \not\equiv 1 \mod \ell_a(q)$ for some $q \mid n$ then
        Set $P[n] = 0$ ;
    else if $n$ fails the strong pseudoprime test then
        Set $P[n] = 0$ ;
```

For complexity, line 1 requires $O(x)$ operations and $O(x \log x)$ space. As noted, Algorithm 1 factors n and computes $\ell_a(n)$ using $O(\log n + \omega(n) \log n / \log \log n)$ multiplications. By applying Lemma 5.6 we calculate the total cost in multiplications of all order computations as

$$O\left(\sum_{q\le x} \log \phi(q) + \omega(\phi(q)) \frac{\log \phi(q)}{\log \log \phi(q)}\right) = O\left(x + \frac{\log x}{\log \log x} \sum_{q \le x} \omega(\phi(q))\right) = O(x) \ .$$

Note that $\sum_{q\le x} 1 = O(x/\log x)$.

Focusing now on line 12, finding the next prime power dividing n and testing a congruence are both single operations. Define $\alpha(n)$ as i if the ith prime power is the smallest obstruction to n being a Fermat pseudoprime, and define $\alpha(n) = \omega(n)$ if n is a Fermat pseudoprime. Then the complexity of the loop at line 12 over all n is $\sum_{n\le x} \alpha(n)$.

We gain insight into this sum if we consider how much work a single prime power $q = p^e$ will create across all n. A prime power q will fail to cross off a non-pseudoprime and thus create an extra operation if $q \mid n$ and $n \equiv 1 \mod \ell_a(q)$. By Proposition 5.4, unless $\gcd(q, \ell_a(q)) \ne 1$ such n are exactly those in the arithmetic progression $n \equiv q \mod q \cdot \ell_a(q)$. An upper bound on the number of operations needed for all n is thus

$$O(x) + \sum_{q \le x} \left\lfloor \frac{x}{q\ell_a(q)} \right\rfloor \le O(x) + \sum_{p \le x} \frac{x}{p\ell_a(p)} + \sum_{\substack{p^r \le x \\ r>1}} \frac{x}{p^r} \ .$$

The first sum is $O(x)$ by [22, Proposition 3] and the second is $O(x)$ by [16, proof of Theorem 430].

Finally, the strong test at line 16 is only performed on Fermat pseudoprimes. Since there are most $x/\sqrt{\exp(\log x \cdot \log \log \log x / \log \log x)}$ Fermat pseudoprimes up to x [23] and the strong test takes $O(\log x)$ operations, the total is $O(x)$ operations. $\square$

An alternate method of sieving for Fermat pseudoprimes is given in [22, Section 7]. Initialize a table of real numbers indexed by integers, and for each prime p add $\log p$ to the entries in the arithmetic progression $n \equiv p \mod p\ell_a(p)$. Then n is a pseudoprime if the total at the end of sieving is $\log n$. An advantage of Algorithm 2 is that all operations are integer arithmetic, obviating the need to track precision.

7 Finding primitive roots and tabulating Fermat liars

Rather than fixing a and looking for n that are a-pseudoprimes, we next fix a composite n and ask for all a that are liars (or witnesses) for n. It is sufficient to restrict the tabulation to $1 \le a < n$, since by definition a is a liar for n if $(a \bmod n)$ is a liar.

Throughout the next two sections we assume n is a positive, odd, composite integer and that the factorizations of both n and $\phi(n)$ are given. Let $n = q_1 \cdots q_k$, where $q_i = p_i^{r_i}$

with p_i prime. Our strategy will be to first show how to find primitive roots modulo q_i, and then demonstrate how to tabulate the elements of $(\mathbb{Z}/n\mathbb{Z})^\times$ of order dividing f, where f is an arbitrary positive integer. This solves the problem of tabulating the set of Fermat liars $F(n)$, but the situation with the set of strong liars is different since $S(n)$ is not generally a subgroup. In the next section we overcome this obstruction through the use of Proposition 5.2. In some sense $S(n)$ is "almost" a subgroup of the group of units.

It is well-known that for prime power q, $(\mathbb{Z}/q\mathbb{Z})^\times$ is cyclic. A generator is called a primitive root. Finding a primitive root is in general a hard problem, but with factorizations of n, $\phi(n)$ on hand there is a straightforward strategy. Take a subset S of $(\mathbb{Z}/q\mathbb{Z})$. For each $s \in S$, compute $\ell_s(q)$ using Algorithm 1. If it equals $\phi(q)$ we know s is a primitive root. The remaining difficulty comes from choosing a small S that contains a primitive root.

We will consider two straightforward methods. One good method is to simply check all integers up to some bound. Define $g(p)$ to be the smallest positive integer which is a primitive root modulo p. Burgess [8] and Wang [30] proved independently that $g(p) = O(p^{1/4+\epsilon})$. The best explicit result is that $g(p) < p^{0.499}$ for $p - 1 \geq \exp(\exp(24))$ [14]. Under the assumption of the Extended Riemann Hypothesis, we have the stronger result that $g(p) = O((\log p)^6)$ [27].

If g is a primitive root modulo a prime p, the following lemma makes it easy to find a primitive root modulo a prime power p^r.

Lemma 7.1. *Let p be an odd prime.*

1. *If $a \in \mathbb{Z}$ is a primitive root modulo p, then one of a or $a+p$ is a primitive root modulo p^2.*
2. *If $a \in \mathbb{Z}$ is a primitive root modulo p^r with $r \geq 2$, then a is a primitive root modulo p^{r+1}.*

Proof. See [3, Section 10.6]. □

An alternative method is to simply choose residues at random. From group theory we know that a cyclic group of order m has $\phi(m)$ generators, and thus the probability of choosing a generator with a single trial is $\phi(m)/m$. We have the following lemma.

Lemma 7.2 ([21], Theorem 2.9)**.** *For all $n \geq 3$,*

$$\phi(n) \geq \frac{(c + o(1))n}{\log \log n}$$

where c is a known absolute constant.

The probability of choosing a primitive root with a single trial is then

$$\frac{\phi(\phi(q))}{\phi(q)} \geq \frac{(c + o(1))\phi(q)}{\log \log(\phi(q))} \frac{1}{\phi(q)} \geq \frac{c + o(1)}{\log \log q} \ .$$

So a primitive root modulo q will be found after an expected $O(\log\log q)$ trials.

With this work we can construct a basis of $(\mathbb{Z}/n\mathbb{Z})^\times$, which we modify to form a basis for the subgroup of elements of order dividing f. Denote that subgroup by H_f.

Definition 7.3. A tuple $\vec{g} = [g_1, \ldots, g_k]$ will be called a basis for a finite abelian group G if every $b \in G$ can be expressed uniquely as $b = \vec{g}^{\vec{t}} = g_1^{t_1} \cdots g_k^{t_k}$ with $0 \le t_i < |g_i|$ for $1 \le i \le k$.

Suppose $G = C_{m_1} \times \cdots \times C_{m_k}$ is a finite abelian group written as a direct product of cyclic groups, with g_i a generator for C_{m_i}. Then we write $\hat{g}_i = (1, 1, \ldots, g_i, \ldots, 1)$ where the non-identity element occurs at the ith place in the tuple. Note that by the theory of direct product groups, $\{\hat{g}_i\}$ forms a basis for G.

Proposition 7.4. *Let $G = C_{m_1} \times C_{m_2} \times \cdots \times C_{m_k}$, where each C_{m_i} is a cyclic group with generator g_i. Then a basis for H_f is $\{\hat{g}_i{}^{m_i/\gcd(m_i,f)}\}$.*

Proof. Let $x_i = m_i/\gcd(m_i, f)$, and note that $(g_i^{x_i})^f = (g_i^{m_i})^{\frac{f}{\gcd(m_i,f)}} = 1$. Thus the set $\{\hat{g}_i{}^{x_i}\}$ at least generates a subgroup of H_f.

Now, focus on a particular cyclic component C_{m_i}. Since g_i has order m_i, $g_i^{x_i}$ has order $\gcd(m_i, f)$. If b is an arbitrary element of C_{m_i} which is also in H_f, then the order of b divides both f and m_i, and hence divides $\gcd(m_i, f)$. But C_{m_i} is cyclic, so there is a unique subgroup of order $\gcd(m_i, f)$ that contains all elements of order dividing $\gcd(m_i, f)$, and as seen a generator is $g_i^{x_i}$.

By group theory, the set $\{\hat{g}_i{}^{x_i}\}$ is thus a basis for H_f. $\square$

Algorithm 3 applies these ideas to compute the structure of the subgroup H_f of $(\mathbb{Z}/n\mathbb{Z})^\times$. We find primitive roots modulo q for each prime power q dividing n, use them to create a basis for $(\mathbb{Z}/n\mathbb{Z})^\times$, then modify them to create a basis for H_f.

Theorem 7.5. *Algorithm 3 correctly computes the structure of the subgroup H_f of $(\mathbb{Z}/n\mathbb{Z})^\times$, given n and $\phi(n)$ in factored form. If primitive root candidates are chosen uniformly at random from the residues modulo p_i, Algorithm 3 has an expected complexity of $O((\log n)^2)$ operations.*

Proof. Line 2 will eventually result in primitive roots being found modulo p_i for each prime dividing n. Then by Lemma 7.1, line 3 finishes with g a primitive root modulo $q_i = p_i^{r_1}$. The Chinese Remainder Theorem is then used to construct a basis for the product group $C_{\phi(q_1)} \times \cdots \times C_{\phi(q_k)}$. By Proposition 7.4, raising these elements to powers $x_i = \phi(q_i)/\gcd(\phi(q_i), f)$ results in a basis for H_f.

For complexity, the length of time needed to find a primitive root depends upon the method used, and whether we assume the Extended Riemann Hypothesis. Let $|S_p|$ be the number of trials before a primitive root modulo p is found. Algorithm 1 has complexity $O(\log p + \omega(p-1)\frac{\log p}{\log\log p})$, while the Chinese Remainder Theorem takes $O((\log n)^2)$ bit

Algorithm 3: Computing the structure of $H_f \subseteq (\mathbb{Z}/n\mathbb{Z})^\times$

```
Input  : Positive integer $n$ in factored form $q_1 \cdot q_2 \cdots q_k$, $\phi(n)$ in factored form
Output: A basis $\{\hat{g}_1, \hat{g}_2, \ldots, \hat{g}_k\}$ for $H_f$ along with $|\hat{g}_i|$
  /* First create $\{\hat{g}_i\}$ as a basis for $(\mathbb{Z}/n\mathbb{Z})^\times$.          */
1 for $1 \leq i \leq k$ do
2     Pick primitive root candidate $g$ and compute $\ell_g(p_i)$ using Algorithm 1 until
      $\ell_g(p_i) = p_i - 1$ ;
3     if $q_i = p_i^{r_i}$ with $r_i \geq 2$ then
4         Use Algorithm 1 to compute $\ell_g(p_i^2)$ ;
5         Set $g \leftarrow g + p_i$ if $\ell_g(p_i^2) \neq p_i(p_i - 1)$ ;
6     Use the Chinese Remainder Theorem to find $\hat{g}_i$ such that $\hat{g}_i \equiv 1 \mod q_j$ for
      $j \neq i$ and $\hat{g}_i \equiv g \mod q_j$ for $j = i$ ;
  /* Now modify $\{\hat{g}_i\}$ to be a basis for $H_f$.          */
7 for $1 \leq i \leq k$ do
8     return $\hat{g}_i \leftarrow \hat{g}_i^{x_i}$ where $x_i = \phi(q_i)/\gcd(\phi(q_i), f)$, along with $|\hat{g}_i| = \gcd(\phi(q_i), f)$;
```

operations or $O(\log n)$ ring operations [5, Corollary 5.5.3]. So the complexity in ring operations of the loop at line 1 is

$$O\left(\sum_{p|n} |S_p| \left(\log p + \omega(p-1)\frac{\log p}{\log\log p}\right) + \log n\right) = O((\log n)^2) + O\left(\sum_{p|n} |S_p| \frac{(\log p)^2}{\log\log p}\right) .$$

If we choose S_p at random, its expected size is $O(\log\log p)$. Note that

$$\sum_{p|n} (\log p)^2 \leq \left(\sum_{p|n} \log p\right)^2 \leq (\log n)^2$$

and so the complexity of the loop at line 1 is an expected $O((\log n)^2)$.

At line 7, $k = O(\log n)$, and for each $1 \leq i \leq k$ we do a gcd, a division, and an exponentiation which takes $O(\log n)$ multiplications. The total cost of this step is thus $O((\log n)^2)$ operations as well. □

If we would prefer a deterministic algorithm, the complexity is greater. Assuming the Extended Riemann Hypothesis, we have

$$\sum_{p|n} |S_p| \frac{(\log p)^2}{\log\log p} \leq \sum_{p|n} (\log p)^8 \leq \left(\sum_{p|n} (\log p)\right)^8 = O((\log n)^8)$$

while unconditionally this is instead

$$\sum_{p|n} |S_p| \frac{(\log p)^2}{\log\log p} = \sum_{p|n} p^{1/4+\epsilon} \leq P(n)^{1/4+\epsilon} + n^{1/8+\epsilon}$$

where $P(n)$ is the largest prime factor of n.

Once the structure of H_f is computed, tabulating the elements involves generating $({\hat{g}_1}^{t_1}, \ldots, {\hat{g}_k}^{t_k})$ for each tuple $\vec{t} = (t_1, \ldots, t_k)$ with $0 \leq t_i < \gcd(\phi(q_i), f)$. If we prefer the elements to be integers, we can update an integer value as we cycle. So if the power of $\hat{g}_i$ is incremented by one, the stored integer is multiplied by $\hat{g}_i$ modulo n.

As a corollary, this gives us a sub-linear time algorithm for tabulating Fermat liars.

Corollary 7.6. *Assume n and $\phi(n)$ are given in factored form. Tabulating $F(n)$ requires at most $O(P(n)^{1/4+\epsilon} + n^{1/8+\epsilon} + |F(n)|)$ operations deterministically, or an expected $O((\log n)^2 + |F(n)|)$ operations if using a randomized method for generating primitive roots modulo p.*

Proof. Apply Algorithm 3 with $f = n - 1$ to compute the structure, then cycle through all the elements. ☐

In fact, since elements of $(\mathbb{Z}/n\mathbb{Z})^\times$ have order dividing $\phi(n)$, we could compute $F(n)$ by instead taking $f = \gcd(n-1, \phi(n))$. This observation is due to Noah Lebowitz-Lockard.

8 Tabulating strong liars

The set of strong liars does not form a group, since if $\nu_2(\ell_a(p)) = \nu_2(\ell_b(p))$ then $\nu_2(\ell_{ab}(p))$ might not have the same value. Instead we explicitly characterize the subset of $F(n)$ with $\nu_2(\ell_a(p))$ equal for all $p \mid n$. After precomputation, tabulating $S(n)$ will require exactly $S(n)$ operations. The author is grateful to an anonymous referee for providing the key idea of this section, improving upon the original algorithm.

We continue to assume n is odd, and that the factorizations of n and $\phi(n)$ are known. Specifically let $n = q_1 \cdots q_k$. Let $\{\hat{g}\}$ be the basis for $F(n)$ computed by Algorithm 3. Recall that $\hat{g}_i = g_i^{x_i}$, where g_i is a primitive root modulo $q_i = p_i^{r_i}$ and $x_i = \phi(q_i)/\gcd(n-1, p_i - 1)$. For a prime p dividing n, decompose $p - 1 = 2^s d$ where d is odd, and let $v(p) = \nu_2(\gcd(n-1, p-1))$. Since the order of $\hat{g}_i$ is $\gcd(n-1, p_i - 1)$, the power of 2 dividing the order is at most $v(p)$. Thus if $a^{n-1} \equiv 1 \mod n$ and $\nu_2(\ell_a(p)) = e$ for all $p \mid n$, we must have $e \leq \min_{p|n} v(p)$.

Proposition 8.1. *The set of $a \in (\mathbb{Z}/p^r\mathbb{Z})^\times$ with $a \in F(n)$ and $\nu_2(\ell_a(p^r)) = \nu_2(\ell_a(p)) = e$ is given by*

$$\{\hat{g}^t \ : \ t \equiv 2^{v(p)-e} \mod 2^{v(p)-e+1}\} \ .$$

if $0 < e \leq v(p)$. If $e = 0$ the set is instead the subgroup generated by $\hat{g}^{2^{v(p)}}$.

Proof. The group we are working with is cyclic, so we again use Proposition 7.4 to characterize the subgroup of elements which are Fermat liars and whose order divides $2^e d$. Since $\hat{g}$ generates a subgroup of order $\gcd(n-1, p-1)$, the subgroup we want is generated by $\hat{g}^y$ where

$$y = \frac{\gcd(n-1,p-1)}{\gcd(\gcd(n-1,p-1),2^e d)} = \frac{\gcd(n-1,p-1)}{\gcd(n-1,2^e d)} \ .$$

By definition d is the odd part of $p-1$ and hence y is a power of 2. Specifically, $y = 2^{v(p)-e}$. Similarly, the subgroup of elements which are Fermat liars and whose order divides $2^{e-1}d$ is generated by $\hat{g}^{2^{v(p)-e+1}}$.

For $e > 0$, $\nu_2(\ell_a(p)) = e$ if and only if $\ell_a(p) \mid 2^e d$ and $\ell_a(p) \nmid 2^{e-1}d$. By the work above, such a are exactly $\hat{g}^t$ where t is a multiple of $2^{v(p)-e}$ but not of $2^{v(p)-e+1}$, i.e. $t \equiv 2^{v(p)-e} \mod 2^{v(p)-e+1}$. If $e = 0$ we simply have the subgroup generated by $\hat{g}^{2^{v(p)}}$. $\square$

A consequence of Proposition 8.1 is that the set of $a \in (\mathbb{Z}/n\mathbb{Z})^\times$ with $\nu_2(\ell_a(p)) = e$ for all $p \mid n$ is exactly $(\hat{g_1}^{t_1}, \ldots, \hat{g_k}^{t_k})$ where tuples $(t_1, \ldots, t_k)$ satisfy $t_i \equiv 2^{v(p_i)-e} \mod 2^{v(p_i)-e+1}$ for $1 \le i \le k$. If we repeat this for all $0 \le e \le \min_{p|n} v(p)$, we will have tabulated all strong liars at a marginal cost of one operation per liar.

Algorithm 4: Tabulating strong liars

```
Input  : odd composite $n$ in factored form $q_1 \cdots q_k$, where $q_i$ is a prime power of
         the prime $p_i$, $\phi(n)$ in factored form
Output: list of strong liars
1 Use Algorithm 3 with $f = n-1$ to construct a basis $\{\hat{g_i}\}$ of $F(n)$ ;
  /* Start with the $e = 0$ subgroup case                                */
2 Compute $\hat{h_i} = \hat{g_i}^{2^{v(p_i)}} \bmod n$ for $1 \le i \le k$;
3 Cycle through all tuples $(\hat{h_1}^{t_1}, \ldots, \hat{h_k}^{t_k})$ where $0 \le t_i < \gcd(n', p_i')$, writing $\prod_i \hat{h_i}^{t_i}$ to
  the list ;
  /* Now for the other cases                                              */
4 for $e = 1$ to $v = \min_i v(p_i)$ do
5     for $i = 1$ to $k$ do
6         precompute $\hat{g_i}^{2^{v(p_i)-e+1}} \bmod n$;
7     Cycle through all tuples $(\hat{g_1}^{t_1}, \ldots, \hat{g_k}^{t_k})$ where $t_i \equiv 2^{v(p_i)-e} \mod 2^{v(p_i)-e+1}$,
      writing $\prod_i \hat{g_i}^{t_i}$ to the list;
```

Theorem 8.2. *Assume n and $\phi(n)$ are given in factored form. Then Algorithm 4 correctly returns $S(n)$. Computing the structure of $F(n)$ takes $O(P(n)^{1/4+\epsilon} + n^{1/8+\epsilon})$ operations deterministically or an expected $O((\log n)^2)$ operations using a randomized algorithm, after which tabulating $S(n)$ requires $O(|S(n)|)$ operations.*

Proof. By Theorem 7.5, Algorithm 3 correctly returns a basis for $F(n)$. Thus every element a constructed during the cycle is a Fermat liar.

By Proposition 8.1, $\hat{g}_i^{t_i}$ with $t_i \equiv 2^{v(p_i)-e} \mod 2^{v(p_i)-e+1}$ creates all $a_i \in (\mathbb{Z}/q_i\mathbb{Z})^\times$ with $\nu_2(\ell_{a_i}(q_i)) = e$ if $e > 0$ and lines 2-3 accomplish the same task in the $e = 0$ case. Then $\nu_2(\ell_a(p)) = e$ for all $p \mid n$ if and only if $a = (a_1, \dots a_k)$ in $(\mathbb{Z}/n\mathbb{Z})^\times$. By Proposition 5.2, $\nu_2(\ell_a(p)) = e$ for some e and for all $p \mid n$ if and only if a is a strong liar for n. If $e > \min_i v(p_i)$, we cannot have $a^{n-1} \equiv 1 \mod n$ and $\nu_2(\ell_a(p_i)) = e$ for some i. Thus by looping over all $0 \leq e \leq \min_i v(p_i)$ we generate all strong liars for n.

Line 1 takes $O(P(n)^{1/4+\epsilon} + n^{1/8+\epsilon})$ operations deterministically or an expected $O((\log n)^2)$ by Theorem 7.5. The precomputation on line 6 requires

$$\sum_{e=1}^{\min_i v(p_i)} \sum_{i=1}^{k} v(p_i) - e + 1 \leq \sum_{i=1}^{k} v(p_i)^2 \leq \sum_{p \mid n} (\log p)^2 = O((\log n)^2)$$

operations. Once the precomputation is complete, incrementing t_i by one step means multiplying by $\hat{g}_i^{2^{v(p_i)-e+1}} \mod n$, which is only a single operation. $\square$

9 Conclusions and acknowledgements

We have shown that tabulating all a-spsp up to x requires linear time in the Big-Oh sense, as does tabulating all strong liars for a given odd composite n.

The literature on prime sieving suggests a host of potential improvements, including wheel datastructures, segmentation to save space, sieving in parallel computing models, and their combinations. Any or all of these might apply to the problems of tabulating pseudoprimes and liars. In particular, using a wheel datastructure would certainly speed up the tabulation of pseudoprimes [25], though analyzing the gain asymptotically is challenging. Perhaps the most promising route to achieving sublinear complexity for tabulating a-spsp lies in analyzing the algorithm laid out in [22].

Many thanks to two anonymous referees for helpful comments and to Jonathan Sorenson for helpful discussions. In particular, the original version of this paper had complexity results with extra $\log \log x$ factors, and the referees suggested improvements for both problems.

References

[1] W. R. Alford, Andrew Granville, and Carl Pomerance, *On the difficulty of finding reliable witnesses*, Algorithmic number theory (Ithaca, NY, 1994), Lecture Notes in Comput. Sci., vol. 877, Springer, Berlin, 1994, pp. 1–16.

[2] ———, *There are infinitely many Carmichael numbers*, Ann. of Math. (2) **139** (1994), no. 3, 703–722.

[3] Tom M. Apostol, *Introduction to analytic number theory*, Springer-Verlag, New York-Heidelberg, 1976, Undergraduate Texts in Mathematics.

[4] Eric Bach, *Explicit bounds for primality testing and related problems*, Math. Comp. **55** (1990), no. 191, 355–380.

[5] Eric Bach and Jeffrey Shallit, *Algorithmic number theory. Vol. 1. Eefficient algorithms*, Foundations of Computing Series, MIT Press, Cambridge, MA, 1996.

[6] Daniel Bleichenbacher, *Efficiency and security of cryptosystems based on number theory*, Ph.D. thesis, Swiss Federal Institute of Technology Zurich, 1996.

[7] Ernest F. Brickell, Daniel M. Gordon, Kevin S. McCurley, and David B. Wilson, *Fast exponentiation with precomputation: algorithms and lower bounds*, Advances in Cryptology – Proceedings of Eurocrypt '92, Lecture Notes in Computer Science, vol. 658, Springer, Berlin, 1992, pp. 200–207.

[8] D. A. Burgess, *On character sums and primitive roots*, Proc. London Math. Soc. (3) **12** (1962), 179–192.

[9] Richard Crandall and Carl Pomerance, *Prime numbers: a computational perspective*, Springer-Verlag, New York, 2001.

[10] Brian Dunten, Julie Jones, and Jonathan Sorenson, *A space-efficient fast prime number sieve*, Inform. Process. Lett. **59** (1996), no. 2, 79–84.

[11] Paul Erdős and Carl Pomerance, *On the number of false witnesses for a composite number*, Math. Comp. **46** (1986), no. 173, 259–279.

[12] Jan Feitsma, *The pseudoprimes below* 2^{64}, 2013, `http://www.janfeitsma.nl/math/psp2/index`.

[13] Andrew Granville and Carl Pomerance, *Two contradictory conjectures concerning Carmichael numbers*, Math. Comp. **71** (2002), no. 238, 883–908.

[14] E. Grosswald, *On Burgess' bound for primitive roots modulo primes and an application to* $\Gamma(p)$, Amer. J. Math. **103** (1981), no. 6, 1171–1183.

[15] H. Halberstam, *On the distribution of additive number-theoretic functions. III*, J. London Math. Soc. **31** (1956), 14–27.

[16] G. H. Hardy and E. M. Wright, *An introduction to the theory of numbers*, sixth ed., Oxford University Press, Oxford, 2008, Revised by D. R. Heath-Brown and J. H. Silverman, with a foreword by Andrew Wiles.

[17] Gerhard Jaeschke, *On strong pseudoprimes to several bases*, Math. Comp. **61** (1993), no. 204, 915–926.

[18] Yupeng Jiang and Yingpu Deng, *Strong pseudoprimes to the first eight prime bases*, Math. Comp. **83** (2014), no. 290, 2915–2924.

[19] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone, *Handbook of applied cryptography*, CRC Press Series on Discrete Mathematics and its Applications, CRC Press, Boca Raton, FL, 1997, with a foreword by Ronald L. Rivest.

[20] Louis Monier, *Evaluation and comparison of two efficient probabilistic primality testing algorithms*, Theoret. Comput. Sci. **12** (1980), no. 1, 97–108.

[21] Hugh L. Montgomery and Robert C. Vaughan, *Multiplicative number theory. I. Classical theory*, Cambridge Studies in Advanced Mathematics, vol. 97, Cambridge University Press, Cambridge, 2007.

[22] Richard G. E. Pinch, *The pseudoprimes up to* 10^{13}, Algorithmic number theory (Leiden, 2000), Lecture Notes in Comput. Sci., vol. 1838, Springer, Berlin, 2000, pp. 459–473.

[23] Carl Pomerance, *On the distribution of pseudoprimes*, Math. Comp. **37** (1981), no. 156, 587–593.

[24] Carl Pomerance, J. L. Selfridge, and Samuel S. Wagstaff, Jr., *The pseudoprimes to* $25 \cdot 10^9$, Math. Comp. **35** (1980), no. 151, 1003–1026.

[25] Paul Pritchard, *Explaining the wheel sieve*, Acta Inform. **17** (1982), no. 4, 477–485.

[26] ———, *Linear prime-number sieves: a family tree*, Sci. Comput. Programming **9** (1987), no. 1, 17–35.

[27] Victor Shoup, *Searching for primitive roots in finite fields*, Math. Comp. **58** (1992), no. 197, 369–380.

[28] Jonathan P. Sorenson and Jonathan Webster, *Strong pseudoprimes to twelve prime bases*, `http://arxiv.org/abs/1509.00864`, 2015.

[29] Andrew V. Sutherland, *Order computations in generic groups*, ProQuest LLC, Ann Arbor, MI, 2007, Thesis (Ph.D.)–Massachusetts Institute of Technology.

[30] Yuan Wang, *On the least primitive root of a prime*, Acta Math. Sinica **9** (1959), 432–441.

[31] Zhenxiang Zhang, *Two kinds of strong pseudoprimes up to* 10^{36}, Math. Comp. **76** (2007), no. 260, 2095–2107.