MATHEMATICS OF COMPUTATION
Volume 89, Number 324, July 2020, Pages 1953–1968
https://doi.org/10.1090/mcom/3501
Article electronically published on January 7, 2020

# TWO ALGORITHMS TO FIND PRIMES IN PATTERNS

JONATHAN P. SORENSON AND JONATHAN WEBSTER

Abstract. Let $k \geq 1$ be an integer, and let $P = (f_1(x), \ldots, f_k(x))$ be $k$ admissible linear polynomials over the integers, or *the pattern.* We present two algorithms that find all integers $x$ where $\max\{f_i(x)\} \leq n$ and all the $f_i(x)$ are prime.

- Our first algorithm takes $O_P(n/(\log\log n)^k)$ arithmetic operations using $O(k\sqrt{n})$ space.
- Our second algorithm takes slightly more time, $O_P(n/(\log\log n)^{k-1})$ arithmetic operations, but uses only $n^{1/c}$ space for $c > 2$ a fixed constant. The correctness of this algorithm is unconditional, but our analysis of its running time depends on two reasonable but unproven conjectures.

We are unaware of any previous complexity results for this problem beyond the use of a prime sieve.

We also implemented several parallel versions of our second algorithm to show it is viable in practice. In particular, we found some new Cunningham chains of length 15, and we found all quadruplet primes up to $10^{17}$.

## 1. Introduction

Mathematicians have long been interested in prime numbers and how they appear in patterns. (See, for example, [12, Ch. A].) In this paper, we are interested in the complexity of the following algorithmic problem:

> Given a pattern and a bound $n$, find all primes $\leq n$ that fit the pattern.

To address this, first we will discuss and define a pattern of primes, then we will look at what is known about the distribution of primes in patterns to see what we can reasonably expect for the complexity of this problem, and finally we will discuss previous work and state our new results.

### 1.1. Prime patterns.

Perhaps the simplest of patterns of primes are the twin primes, which satisfy the pattern $(x, x+2)$ where both $x$ and $x+2$ are prime. Examples include 59,61 and 101,103.

We can, of course, generalize this to larger patterns. For example, prime quadruplets have the form $(x, x+2, x+6, x+8)$, and examples include 11,13,17,19 and 1481,1483,1487,1489.

Larger patterns of primes of this type are called *prime k-tuples.* If the $k$-tuple has the smallest possible difference between its first and last primes (its *diameter*), it is also called a *prime constellation.* The pattern $(x, x+2, x+6, x+8)$ is a

Received by the editor July 23, 2018, and, in revised form, February 14, 2019, and October 31, 2019.

2010 *Mathematics Subject Classification.* Primary 11A41, 11Y11, 11Y16, 68Q25.

A preliminary version of this work was presented as a poster at the ANTS XIII conference in Madison, Wisconsin, in July of 2018.

constellation, as 8 is the smallest possible diameter for a pattern of length 4. There are two constellations of length 3: $(x, x+2, x+6)$ and $(x, x+4, x+6)$. See, for example, [7, §1.2.2] or [25, Ch. 3].

Sophie Germain studied the pattern $(x, 2x+1)$, which was later generalized to *Cunningham chains* of two kinds. Chains of the *first kind* have the pattern $(x, 2x+1, 4x+3, 8x+7, \ldots)$, and chains of the *second kind* have the pattern $(x, 2x-1, 4x-3, 8x-7, \ldots)$.

Chernick [6] showed that any prime pattern of the form $(6x+1, 12x+1, 18x+1)$ gives a Carmichael number composed of the product of these three primes.

Let $k > 0$ be an integer. A *prime pattern* of size $k$ is a list of $k$ linear polynomials over the integers with positive leading coefficients, $(f_1(x), \ldots, f_k(x))$. A pattern of size $k$ is *admissible* if for every prime $p \leq k$, there is an integer $x$ such that $p$ does not divide any of the $f_i(x)$. For an algorithm to test for pattern admissibility, see [25, pp. 62–63].

We restrict our notion of pattern to linear polynomials in this paper.

1.2. **The distribution of primes in patterns.** The unproven twin prime conjecture states there are infinitely many twin primes. Yitang Zhang [33] recently showed that there is a positive integer $h$ such that the pattern $(x, x+h)$ is satisfied by infinitely many primes.

The Hardy-Littlewood $k$-tuple conjecture [14] implies that each pattern, with leading coefficients of 1, that is *admissible* will be satisfied by primes infinitely often. Further, the conjecture implies that the number of primes $\leq n$ in such a pattern of length $k$ is roughly proportional to $n/(\log n)^k$.

For twin primes, then, the Hardy-Littlewood conjecture gives an estimate of

$$2C_2 \frac{n}{(\log n)^2}$$

for the number of twin primes $\leq n$, where $C_2 \approx 0.6601\ldots$ is the twin primes constant. Brun's theorem gives an $O(n/(\log n)^2)$ upper bound for the number of twin primes $\leq n$. For every $k$-tuple with $k \geq 2$, there is a corresponding conjectured constant of proportionality that depends on the pattern and on $k$, generalizing the twin primes constant.

Dickson's conjecture [8] states that there are infinitely many primes satisfying any fixed admissible pattern, even with leading coefficients $> 1$. Thus, it applies to fixed-length Cunningham chains as well.

We have the following upper bound, due to Halberstam and Richert [13, Thm. 2.4].

**Lemma 1.** *Given an admissible pattern $(f_1(x), \ldots, f_k(x))$ of length $k$, the number of integers $x$ such that the $f_i(x)$ are all simultaneously prime and $\max\{f_i(x)\} \leq n$ is $O(n/(\log n)^k)$.*

Here the implied constant of the big-$O$ can depend on $k$ but does not depend on the coefficients of the $f_i$.

1.3. **Previous work.** We can obtain a rather straightforward analysis by simply finding *all* primes $\leq n$ and then scanning to see how many tuples match the pattern. Note that it is not necessary to write all the primes down; we can scan them in small batches as we go. Since the scan phase is fast, the bottleneck would be finding the primes using a sieve. The Atkin-Bernstein sieve [2] does this using at most $O(n/\log\log n)$ arithmetic operations and $\sqrt{n}$ space. Galway [10] showed how to reduce space further to roughly $n^{1/3}$, but this version could not use the wheel data structure and requires $O(n)$ arithmetic operations. See also [16].

We follow the convention from the analysis of prime sieves of not charging for the space used by the output of the algorithm. Note that if we charged for the space of the output, we could not expect to do better than $n/(\log n)^{k-1}$ bits in general.

A more specialized algorithm that searches for only the primes in the pattern and not all the primes should do much better. Günter Löh [20] and Tony Forbes [9] described algorithms to do exactly this but gave no runtime analysis. It seems likely their algorithms are faster than $O(n/\log\log n)$, but without an analysis, we don't know if this is true or not. Note that Forbes outlined an odometer-like data structure that seems to be similar to the wheel data structure we employ.

Of course, by the prime number theorem, there are only about $n/\log n$ primes $\leq n$, so the current best prime sieves use $\log n/\log\log n$ arithmetic operations per prime on average. We do not know if anything smaller is possible. Applying this average cost to the results of Lemma 1, we can hope for an algorithm that takes $O(n/(\log\log n)^k)$ arithmetic operations to find all primes $\leq n$ in a fixed pattern of length $k$.

In essence, this is what we prove as our main result.

### 1.4. New results.

Our contribution is the following.

**Theorem 1.** *There is an algorithm that when given a list of $k$ distinct linear polynomials over the integers, with positive leading coefficients, $P = (f_1(x), \ldots, f_k(x))$ (the pattern), and a search bound $n$ finds all integers $x$ such that $\max\{f_i(x)\} \leq n$ and all the $f_i(x)$ are prime. This algorithm uses at most $O_P(n/(\log\log n)^k)$ arithmetic operations and $O(k\sqrt{n})$ bits of space.*

This algorithm extends the Atkin-Bernstein prime sieve with our space-saving wheel sieve. See [28–30].

The $\sqrt{n}$ space needed by this algorithm limits its practicality. By replacing the Atkin-Bernstein sieve with the sieve of Eratosthenes combined with prime tests, we can greatly reduce the need for space.

**Theorem 2.** *Let $c > 2$. There is an algorithm that when given a list of $k > 2$ distinct linear polynomials over the integers, with positive leading coefficients, $P = (f_1(x), \ldots, f_k(x))$ (the pattern), and a search bound $n$ finds all integers $x$ such that $\max\{f_i(x)\} \leq n$ and all the $f_i(x)$ are prime. This algorithm uses at most $O_P(n/(\log\log n)^{k-1})$ arithmetic operations and $n^{1/c}$ bits of space, under the assumption of Conjectures 1 and 2 (see below). Correctness of the output is unconditional.*

If $k > 6$, then Conjecture 1 is not needed. We use the sieve of Eratosthenes with primes up to a bound $B = n^{1/c}$, after which we apply a base-2 pseudoprime test and then a version of the AKS prime test [1] that was improved to take $(\log n)^{6+o(1)}$ time [19]. Our goal here is to balance the cost of sieving with the cost of prime testing. For the range $2 < k \leq 6$, to keep the cost of prime testing low enough, we replace the AKS prime test with the pseudosquares prime test of Lukes, Patterson, and Williams [21]. This prime test takes only $O((\log n)^2)$ time under the condition of an unproven but reasonable conjecture on the distribution of pseudosquares due to Bach and Huelsbergen [3]. Note that the correctness of our algorithm does not rely on any unproven conjectures.

The *pseudosquare* $L_p$ is the smallest positive integer that is not a square, is 1 mod 8, and is a quadratic residue modulo every odd prime $q \le p$.

**Conjecture 1** (Bach and Huelsbergen [3])**.** *Let $L_p$ be the largest pseudosquare $\le n$. Then $p = O(\log n \log\log n)$.*

Our second conjecture is needed to bound the cost of prime testing after sieving out by primes $\le y$ in an arithmetic progression. Let $p(n)$ denote the smallest prime divisor of the integer $n$.

**Conjecture 2.** *Let $a, b$ be positive integers with* $\gcd(a,b) = 1$*. Then*

$$\#\{n \le x, n \equiv a \bmod b, p(n) > y\} \ll \frac{x}{b} \prod_{\substack{p \le y \\ \gcd(p,b)=1}} \left(1 - \frac{1}{p}\right).$$

This would be a theorem if $b \le \sqrt{x}$; see [13, (1.7), (1.8); 32].

We performed a few computations with this second version of our algorithm to show its practicality. A couple of these computational results are new.

The rest of this paper is organized as follows. In §2 we present our proof of Theorem 1, including our model of computation in §2.1, a description of our first algorithm in §2.2, and its running time analysis in §2.3. In §3 we discuss our second algorithm in §3.1 and its analysis in §3.2, thereby proving Theorem 2. We present our computational results in §4, including our work on twin primes in §4.1, our work on prime quadruplets in §4.2, and our results on Cunningham chains in §4.3. We wrap up with a discussion of possible future work in §5.

## 2. Theory

### 2.1. **Model of computation.**

Our model of computation is a standard random access machine with infinite, direct-access memory. Memory can be addressed at the bit level or at the word level, and the word size is $\Theta(\log n)$ bits if $n$ is the input. Arithmetic operations on integers of $O(\log n)$ bits take constant time, as do memory/array accesses, comparisons, and other basic operations.

We count space used in bits, and we do not include the size of the output.

### 2.2. **Our first algorithm.**

In this section we present the version of our algorithm with the smallest running time; we perform the analysis in the next section.

The input to the algorithm is the search bound $n$ and the pattern, which consists of the value of $k$ and the list of linear polynomials $(f_1(x), \dots, f_k(x))$. We write $a_i$ for the multiplier and $b_i$ for the offset for each form $f_i$. For simplicity, we often assume that $a_1 = 1$ and $b_1 = 0$, but this convenience is not required to obtain our complexity bound. So, for example, for Cunningham chains of the first kind we would have $a_1 = 1, a_2 = 2, a_3 = 4, \dots, a_k = 2^{k-1}$ and $b_1 = 0, b_2 = 1, b_3 = 3, \dots, b_k = a_k - 1$.

(1) We begin by finding the list of primes up to $\lfloor\sqrt{n}\rfloor$ and choosing a subset to be the *wheel primes* $\mathcal{W}$. Define $W := \prod_{p \in \mathcal{W}} p$. Generally, we put all primes up to a bound $y$ into $\mathcal{W}$, with $y$ maximal such that $W \le \sqrt{n}$. Then by the prime number theorem [15] we have

$$(1/2)\log n \sim \log W = \sum_{p \le y} \log p \sim y.$$

This implies that $\sqrt{n}/\log n \ll W \le \sqrt{n}$.

In practice, if there is a prime $\leq y$ that provides poor filtering for the pattern, we consider dropping it from $\mathcal{W}$ and increasing $y$ to include another prime.

We must also check all primes $\leq y$ to see if they participate in the prime pattern.

(2) Next, we construct the wheel data structure so that it will generate all acceptable residues modulo $W$.

The data structure is initialized with a list of pairwise coprime moduli and for each such modulus $m$, a list of acceptable residues mod $m$, encoded as a bit vector of length $m$. The wheel modulus $W$ is then the product of these moduli. Once initialized, the wheel has the ability to enumerate suitable residues modulo $W$ in amortized constant time per residue. The residues do not appear in any particular order.

Therefore, for each prime $p \in \mathcal{W}$ we compute a bit vector (`ones[]`) that encodes the list of acceptable residues. For any integral $x$, we want $f_i(x) = a_i x + b_i$ to not be divisible by $p$. So if $p$ divides $a_i x + b_i$ or, equivalently, if $p$ does not divide $a_i$ and so $x \equiv -b_i \cdot a_i^{-1} \bmod p$, then bit position $x$ must be a zero.

```
vector<bool> ones($p$,1); // length $p$, all 1s to start
for($i = 0$; $i < k$; $i = i + 1$)
      if($a_i \bmod p \neq 0$)
            ones[ (-$b_i \cdot a_i^{-1} \bmod p$) ]= 0;
```

Continuing the example above, for Cunningham chains of the first kind, $p = 3$ gives the vector 001, and $p = 7$ gives the vector 0010111.

We then construct the wheel data structure as described in [28, §4].

(3) For each residue $r \bmod W$ generated by the wheel, we sieve $k$ arithmetic progressions for primes up to $n$, $f_i(r) = a_i r + b_i \bmod W$, or $(a_i r + b_i) + j \cdot a_i W$ for $j = 0, \ldots, \lfloor n/(a_i W) \rfloor$. We do this using the Atkin-Bernstein sieve. (See [2, §5] for how to use the sieve to find primes in an arithmetic progression.)

For each $r$, this yields $k$ bit vectors of length $\leq n/W$ which are combined using a bitwise AND operation to obtain the bit positions for where the pattern is satisfied by primes.

**Example.** Let us find the prime quadruplets ($k = 4$, $P = (x, x+2, x+6, x+8)$) up to $n = 1000$ using a wheel of size $2 \cdot 3 \cdot 5 \cdot 7 = 210$.

We generate just the one quadruplet $5, 7, 11, 13$ that contains wheel primes.

The only acceptable residue mod 2 is 1, for 3 it is 2 (if $x \equiv 1 \bmod 3$, then $x + 2$ is divisible by 3), giving 5 mod 6, and for 5 it is 1, giving 11 mod 30. For 7 there are three acceptable residues, $2, 3, 4$, giving us $11, 101, 191 \bmod 210$ by the Chinese remainder theorem.

For $r = 11 \bmod 210$, we sieve the progression $11 + j \cdot 210$ for primes, getting the bit vector 10110; 11, 431, and 641 are prime, 221 and 851 are not. We then sieve $r + 2 = 13 \bmod 210$ for primes, getting 11111; they are all prime. Sieving $r + 6 = 17 \bmod 210$ gives 11011, and $r + 8 \bmod 210$ gives 11101.

| | | | | |
|---|---|---|---|---|
| 11 | ~~221~~ | 431 | 641 | ~~851~~ |
| 13 | 223 | 433 | 643 | 853 |
| 17 | 227 | ~~437~~ | 647 | 857 |
| 19 | 229 | 439 | ~~649~~ | 859 |

Bitwise AND-ing these four vectors together gives 10000. The only quadruplet we find is $(11, 13, 17, 19)$.

Doing the same for $r = 101$, we get the following:

| | | | | |
|---|---|---|---|---|
| 101 | 311 | 521 | ~~731~~ | 941 |
| 103 | 313 | 523 | 733 | ~~943~~ |
| 107 | 317 | ~~527~~ | ~~737~~ | 947 |
| 109 | ~~319~~ | ~~529~~ | 739 | ~~949~~ |

which gives us the quadruplet $(101, 103, 107, 109)$.

Finally for $r = 191$ we get

| | | | |
|---|---|---|---|
| 191 | 401 | ~~611~~ | 821 |
| 193 | ~~403~~ | 613 | 823 |
| 197 | ~~407~~ | 617 | 827 |
| 199 | 409 | 619 | 829 |

and we get two quadruplets, $(191, 193, 197, 199)$ and $(821, 823, 827, 829)$.

### 2.3. **Complexity analysis.**

We'll look at the cost for each step of the algorithm above.

(1) We can use the Atkin-Bernstein sieve to find the primes up to $\sqrt{n}$ in $O(\sqrt{n}/\log\log n)$ arithmetic operations using $n^{1/4}$ space.

(2) Recall that the largest wheel prime is roughly $(1/2)\log n$. Constructing the bit vector `ones[]` for one prime takes $O(\log n)$ time to initially write all ones and then $O(k)$ time to mark the zeros. Summing over all wheel primes gives $O((\log n)^2/\log\log n)$ operations.

From [28, Thm. 4.1] the total cost to build the wheel is $O((\log n)^3)$ operations, and it occupies $O((\log n)^3/\log\log n)$ space.

(3) The Atkin-Bernstein sieve finds all primes in an arithmetic progression in an interval of size $\sqrt{n}$ or larger in time linear in the length of the interval using space proportional to $\sqrt{n}$ [2, §5]. Therefore, sieving for primes takes $O(n/W)$ operations for each of the $k$ residue classes $f_i(r) \bmod W$, for a total of $O(kn/W)$. The cost to generate each value of $r$ using the wheel is negligible in comparison. The space used is $O(k\sqrt{n})$ bits.

Next we show that the total number of residues is roughly asymptotic to $W/(\log\log n)^k$. For a pattern $P$ of size $k$, all but finitely many primes $p$ will have $p-k$ possible residues. Let $b(P)$ be a constant, depending on the pattern $P$, such that all primes larger than $b(P)$ have $p-k$ residues. Recall that $y$ is a bound for the largest prime in the wheel. Then the total number of residues $r \bmod W$ will be asymptotically bounded by

$$\begin{aligned} \prod_{p\le b(P)} p \cdot \prod_{b(P)<p\le y} (p-k) \quad &\le \quad W \prod_{b(P)<p\le y} \frac{p-k}{p} \\ &= \quad W \prod_{b(P)<p\le y} \left(1-\frac{k}{p}\right). \end{aligned}$$

By Bernoulli's inequality we have $1 - k/p \leq (1-1/p)^k$, so that

$$\begin{aligned} W \prod_{b(P)<p\leq y} \left(1-\frac{k}{p}\right) &\leq W \prod_{b(P)<p\leq y} \left(1-\frac{1}{p}\right)^k \\ &= W \prod_{p\leq b(P)} \left(1-\frac{1}{p}\right)^{-k} \cdot \prod_{p\leq y} \left(1-\frac{1}{p}\right)^k . \end{aligned}$$

The first product is bounded by a constant that depends on $P$, which we refer to as $c(P)$ going forward. If $y \geq 4$, then by [26, (3.26)] we know that

$$\prod_{p\leq y} \left(1-\frac{1}{p}\right) < \frac{1}{\log y}.$$

As shown above, if $y$ is the largest prime in $\mathcal{W}$, we have $y \sim (1/2)\log n$. Asymptotically, $y \geq (1/10)\log n$ is a very safe underestimate. Pulling this all together, we obtain the bound

$$c(P)\frac{W}{(\log\log n)^k}$$

for the total number of residues $r \bmod W$.

Multiplying the sieving cost by the number of residues gives

$$O(c(P)n/(\log\log n)^k)$$

operations.

We have proved Theorem 1.

It is possible to pin down $c(P)$, the constant that depends on $P$ (and $k$). We can use [26, (3.30)] and, assuming $b(P) \geq 3$ (note that $b(P) \geq k$), we have

$$\prod_{p\leq b(P)} \frac{p}{p-1} < 3.26 \log b(P).$$

Bringing in the factor of $k$ introduced in the sieving, we have

$$c(P) \leq k(3.26\log b(P))^k.$$

It remains to get a bound for $b(P)$. Set $A := \max\{|a_j|, |b_j|\}$, an upper bound on all the coefficients in the pattern. Set $F(x) := \prod_{i=1}^k f_i(x) = \prod_{i=1}^k (a_i x + b_i)$. For a prime $p$, if $F$ has repeated roots modulo $p$, then $p \leq b(P)$. But then $a_i x + b_i \equiv a_j x + b_j \bmod p$ for some $i, j$ with $i \neq j$, which means $p$ divides $a_i b_j - a_j b_i$, and hence $p \leq 2A^2$. Thus $b(P) \leq 2A^2$, and we have the bound

$$c(P) \leq k(3.26\log(2A^2))^k.$$

For specific patterns, the constant can be computed explicitly, typically giving much better results than this. In particular, we assumed every prime $\leq b(P)$ contributed all its residues (which happens if a prime divides all the $a_i$'s), which is unusual.

As an example, for our pattern $P = (x, x+2, x+6, x+8)$, we have $b(P) = 3$, and our constant would be

$$4 \cdot \left(\frac{1/2}{(1-1/2)^4}\right)\left(\frac{1/3}{(1-1/3)^4}\right) = 2\cdot 3^3 = 54.$$

Here the $1/2$ and $1/3$ multipliers are included because 2 and 3 have only one residue each instead of 2 and 3, respectively.

Note that one of the anonymous referees deserves the primary credit for this bound on $c(P)$.

## 3. Practice

The primary difficulty in reaching large values of $n$ with our first algorithm is the amount of space it requires. One way to address this is to create a larger wheel, sieve more but shorter arithmetic progressions for primes, and rely less on sieving and more on primality tests (in the style of [28]) when searching for $k$-tuples.

We use the sieve of Eratosthenes instead of the Atkin-Bernstein sieve for the arithmetic progressions, and this is the source of the $\log \log n$ factor slowdown. The gains here are less space needed by a factor of $k$, and the effective trial division performs a quantifiable filtering out of non-primes.

Instead of sieving by all primes up to $\sqrt{n}$, we sieve only by primes up to a bound $B := n^{1/c}$ for a constant $c > 2$. In practice, we choose $B$ so everything, including space for the wheel data structure, the bit vector for sieving, and the list of primes $\leq B$, fits in CPU cache. We then use a base-2 pseudoprime test, followed by a prime test as needed. For smaller $k$, we use the pseudosquares prime test of Lukes, Patterson, and Williams [21], which is fast and deterministic, assuming a sufficient table of pseudosquares is available. Importantly, it takes advantage of the trial division effect of the sieve of Eratosthenes. For larger $k$, we can simply use the AKS prime test [1].

This change means we can get by with only $O(B)$ space. Choosing $B$ larger or smaller gives a tradeoff between the cost of sieving and the cost of performing base-2 pseudoprime tests.

### 3.1. **Our second algorithm.**

(1) Choose a constant $c > 2$, and then set $B := 2^{\lfloor (1/c) \log_2 n \rfloor}$, a power of 2 near $n^{1/c}$. We begin by finding the list of primes up to $B$ and dividing them into the two sets $\mathcal{W}$ and $\mathcal{S}$. Small primes go into $\mathcal{W}$ and the remainder go in $\mathcal{S}$. We want $W := \prod_{p \in \mathcal{W}} p$ to be as large as possible with the constraint that $W \leq n/B$. This implies that the largest prime in $\mathcal{W}$ will be roughly $\log n$.

(2) If $k \leq 6$, we will need to perform the pseudosquares prime test, so in preparation, find all pseudosquares $L_p \leq n/B$ (see [21]).

(3) Next, as before, we construct the wheel data structure so that it will generate all possible correct residues modulo $W$.

(4) For each residue $r \bmod W$ generated by the wheel, we construct a bit vector `v[]` of length $n/W$. Each vector position `v[`$j$`]`, for $j = 0, \ldots, \lfloor W/n \rfloor$, represents $x(j) = r + j \cdot W$ for the $k$-tuple $(f_1(x(j)), f_2(x(j)), \ldots, f_k(x(j)))$. We initialize `v[`$j$`]`$= 1$ but clear it to 0 if we find a prime $p \in \mathcal{S}$ where $p \mid f_i(x(j))$ for some $i$.

```
for( $p \in \mathcal{S}$)
  $winv := W^{-1} \bmod p$;
  for($i := 0$; $i < k$; $i := i + 1$)
    $j := winv \cdot (-b_i a_i^{-1} - r) \bmod p$;
    while($j < n/W$)
      v[$j$]$:= 0$;
      $j := j + p$;
```

Once this sieving is complete, the only integers $j$ with `v[`$j$`]`$= 1$ that remain satisfy the property that all the $f_i(x(j))$ have no prime divisors less than $B$.

(5) For each such $x(j)$ remaining (that is, `v[`$j$`]`$= 1$), we first do a base-2 strong pseudoprime test on $f_1(x(j))$. If it fails, we cross it off (set `v[`$j$`]`$= 0$). If it passes, we try $f_2(x)$ and so forth, keeping `v[`$j$`]`$= 1$ only if all $k$ values $f_i(x(j))$ pass the pseudoprime test. We then perform a full prime test on the $f_i(x(j))$ for all $i$. If $k \leq 6$, we use the Lukes, Patterson, and Williams pseudosquares prime test [21] as done in [28]. For larger $k$, we use the AKS prime test [1]. (This is for the purposes of the theorem; in practice, the pseudosquares prime test is faster, so we use that instead.) If all the $f_i(x(j))$ pass the prime tests, the corresponding $k$-tuple is written for output.

This version of the algorithm works best for $k \geq 4$. When $k \leq 3$, the prime tests become the runtime bottleneck, and so we recommend using $B = \sqrt{n}$ so that the base-2 pseudoprime tests and the pseudosquares prime test are not needed, as the sieving will leave only primes.

**Example.** We use the same example as above, finding prime quadruplets up to 5000, which uses the pattern $(x, x+2, x+6, x+8)$. We go a bit higher this time to illustrate the sieving. Recall that our wheel uses the primes $2 \cdot 3 \cdot 5 \cdot 7$ and generates the three residues $11, 101, 191$ modulo 210.

We will use $B = 20$ as our sieve bound, so $\mathcal{S} = \{11, 13, 17, 19\}$. As with the wheel primes, quadruplets that include sieve primes must be generated separately, so we output the quadruplet $(11, 13, 17, 19)$ at this point.

For $r = 11$, we have our progression

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 11 | 221 | 431 | 641 | 851 | 1061 | 1271 | 1481 | 1691 | 1901 |
| 2111 | 2321 | 2531 | 2741 | 2951 | 3161 | 3371 | 3581 | 3791 | 4001 |
| 4211 | 4421 | 4631 | 4841 | | | | | | |

For $p = 11$, we cross off integers that are $0, 3, 5, 9$ modulo 11. This takes four passes: on the first pass we remove $11, 2321, 4631$, which are 0 mod 11; on the second pass we remove $641, 2951$, which are 3 mod 11 (for example $649 = 11 \cdot 59$); the third removes $1061, 3371$, which are 5 mod 11; and the fourth removes $1901, 4211$, which are 9 mod 11. Observe that for a given residue modulo 11, the numbers to cross off are exactly 11 spaces apart.

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| ~~11~~ | 221 | 431 | ~~641~~ | 851 | ~~1061~~ | 1271 | 1481 | 1691 | ~~1901~~ |
| 2111 | ~~2321~~ | 2531 | 2741 | ~~2951~~ | 3161 | ~~3371~~ | 3581 | 3791 | 4001 |
| ~~4211~~ | 4421 | ~~4631~~ | 4841 | | | | | | |

For $p = 13$, we cross off integers that are $0, 5, 7, 11$ modulo 13. We remove $221, 2951$ for 0 mod 13, $2111, 4841$ for 5 mod 13, 2321 for 7 mod 13, and $11, 2741$ for 11 mod 13.

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| ~~11~~ | ~~221~~ | 431 | ~~641~~ | 851 | ~~1061~~ | 1271 | 1481 | 1691 | ~~1901~~ |
| ~~2111~~ | ~~2321~~ | 2531 | ~~2741~~ | ~~2951~~ | 3161 | ~~3371~~ | 3581 | 3791 | 4001 |
| ~~4211~~ | 4421 | ~~4631~~ | ~~4841~~ | | | | | | |

For $p = 17$, we cross off integers that are $0, 9, 11, 15$ modulo 17. We remove $221, 3791$ for 0 mod 17, 2321 for 9 mod 17, 2531 for 15 mod 17, and $11, 3581$ for 11 mod 17.

~~11~~ ~~221~~ 431 ~~641~~ 851 ~~1061~~ 1271 1481 1691 ~~1901~~
~~2111~~ ~~2321~~ ~~2531~~ ~~2741~~ ~~2951~~ 3161 ~~3371~~ ~~3581~~ ~~3791~~ 4001
~~4211~~ 4421 ~~4631~~ ~~4841~~

For $p = 19$, we cross off integers that are $0, 11, 13, 17$ modulo 19. We remove $11, 4001$ for 11 mod 19, $431, 4421$ for 13 mod 19, 1271 for 17 mod 19, and 1691 for 0 mod 19.

~~11~~ ~~221~~ ~~431~~ ~~641~~ 851 ~~1061~~ ~~1271~~ 1481 ~~1691~~ ~~1901~~
~~2111~~ ~~2321~~ ~~2531~~ ~~2741~~ ~~2951~~ 3161 ~~3371~~ ~~3581~~ ~~3791~~ ~~4001~~
~~4211~~ ~~4421~~ ~~4631~~ ~~4841~~

At this point we perform base-2 strong pseudoprime tests, followed by prime tests as needed. Here 851 and 3161 are not prime, and 1481 leads to the quadruplet $(1481, 1483, 1487, 1489)$.

This is then repeated with $r = 101, 191$.

3.2. **Complexity analysis.** Finding the primes up to $B$ takes $O(B)$ time using $O(\sqrt{B})$ space, well within our bounds. See [28] for a sublinear time algorithm to find all needed pseudosquares. In practice, all pseudosquares up to $10^{25}$ are known [29]. The cost in time and space to build the wheel is, up to a constant factor, the same. So we now focus on steps (4) and (5).

As shown above, the number of residues to check $\bmod W$ is $O_P(W/(\log\log n)^k)$. The time to sieve each interval of length $n/W$ using primes up to $B$ is at most proportional to

$$\sum_{p\le B} \frac{kn}{pW} \sim \frac{kn\log\log B}{W} \sim \frac{kn\log\log n}{W}.$$

Here the multiplier $k$ is required because we cross off $k$ residues modulo most of the primes $p \le B$. That said, this multiple of $k$ can be absorbed into the implied constant that depends on the pattern $P$, $c(P)$, from earlier.

At this point we make use of Conjecture 2 to bound the number of integers free from prime divisors $\le B$ in an arithmetic progression.

With this assumption in hand, by Mertens's theorem, an average of at most

$$\frac{n}{W}\prod_{y<p\le B}\left(1-\frac{1}{p}\right) \quad \ll \quad \frac{n\log y}{W\log B} \quad \sim \quad \frac{n\log\log n}{W\log n}$$

vector locations remain to be prime tested. (Note that we cannot make any assumptions about the relative independence of the primality of the $f_i(x)$ values for different $i$, and so we cannot use a $(1 - k/p)$ factor here.)

A single base-2 strong pseudoprime test takes $O(\log n)$ operations to perform, for a total cost proportional to

$$\frac{kn\log\log n}{W\log n}\log n \sim \frac{kn\log\log n}{W}$$

arithmetic operations to do the base-2 strong pseudoprime tests for each value of $r \bmod W$. This matches the sieving cost of $O_P(n\log\log n/W)$ from above. (Note that if we deliberately choose a larger value for $B$, the increased sieving will decrease the number of pseudoprime tests needed. This tradeoff can be used to fine-tune the running time of the algorithm.)

Thus, the total cost for sieving and base-2 pseudoprime tests is

$$O_P\left(\frac{n}{(\log\log n)^{k-1}}\right),$$

which we obtain by multiplying by the number of residues $O_P(W/(\log\log n)^k)$.

Next we need to count integers that pass the base-2 strong pseudoprime test. Such integers are either prime or composite base-2 pseudoprimes. We switch to counting across all residues $r \bmod W$ to obtain an overall bound.

Lemma 1 tells us that at most $O(n/(\log n)^k)$ integers are prime that fit the pattern, so this is an upper bound on primes that pass the base-2 pseudoprime test.

Pomerance [24] showed that the number of composite base-2 pseudoprimes $\le n$ is bounded by

$$ne^{-\sqrt{\frac{\log n \log\log\log n}{\log\log n}}} \ll \frac{n}{(\log n)^{k+1}},$$

which is negligible. This plus the bound for primes above gives us the $O(n/(\log n)^k)$ bound we desire for all integers that pass the base-2 pseudoprime test.

Next, to bound the cost of prime tests, we have two cases: $k > 6$ or $2 < k \le 6$.

For $k > 6$, we use the AKS prime test [1] (improved in [19]) which takes time $O((\log n)^{6+o(1)})$. The cost of applying the AKS prime test to all the integers $f_i(x)$ after they all pass a base-2 pseudoprime test is at most proportional to

$$k \cdot (\log n)^{6+o(1)} \cdot \frac{n}{(\log n)^k} \ll \frac{kn}{(\log n)^{k-6+o(1)}},$$

which is $o(kn/(\log\log n)^k)$ for $k > 6$.

Note that when $k$ is large, in practice we might only do the base-2 pseudoprime tests and then run full prime tests on the output afterwards, since the amount of output will be rather small.

For $2 < k \le 6$, Conjecture 1 implies that the pseudosquares prime test takes $O((\log n)^2)$ arithmetic operations to test integers $\le n$ for primality, given a table of pseudosquares $\le n$. If $n$ has no prime divisors below $B$, then pseudosquares up to $n/B$ suffice. See [21, 28].

So, under the assumption of Conjecture 1, the cost of applying the pseudosquares prime test to all the integers $f_i(x)$ after they all pass a base-2 pseudoprime test is at most proportional to

$$k \cdot (\log n)^2 \cdot \frac{n}{(\log n)^k} \ll \frac{kn}{(\log n)^{k-2}},$$

and this is $o(kn/(\log\log n)^k)$ for $k > 2$.

The space used is dominated by the length of the sieve intervals and the space needed to store the primes in $\mathcal{S}$, which is $O(B)$ bits.

This completes the proof of Theorem 2.

## 4. Computations

As mentioned previously, we implemented several versions of our second algorithm to see what we could compute. We looked for new computational records that were within reach of our university's nice but aging hardware. Below we discuss some of the results of those computations. Some of the implementation details are specific to a particular computation. Here are four remarks about implementation details that these computations had in common.

(1) We wrote our programs in C++ using the GNU compiler under Linux. GMP was used for multiprecision arithmetic when necessary. Note that it was fairly easy to write the code such that GMP was needed only on occasion and for prime tests.

(2) We used MPI and ran our code on Butler University's cluster Big Dawg. This machine has 16 compute nodes with 12 cores (2 CPUS), each at optimal capacity. Our average utilization was around 150 of the 192 cores due to compute nodes going down from time to time. The CPU is the Intel Xeon CPU E5-2630 0 @ 2.30GHz with 15 MB cache, with 6 cores per CPU.

To parallelize the algorithm, we striped on the residues $r \bmod W$. In other words, all $\nu$ processors stepped through all the $r \bmod W$ residues but only sieved every $\nu$th residue for primes. This meant there was very little communication overhead, except for when periodic checkpoints were done, about every 15-30 minutes.

(3) We usually chose our wheel size ($W$) and sieve intervals so that the size of each interval ($n/W \approx B$) was at most a few megabytes so that it would fit in the CPU cache. We used a `vector<bool>`, which packs bits.

(4) For larger values of $k$, we observed that when sieving by smaller primes $p$ by each of the $k$ residues, we might find that almost all the bits of the current interval were cleared long before we reached the sieving limit $B$, so we created a simple early-abort strategy that was able to save time.

The very few remaining bits were tested with the base-2 strong pseudoprime test even though we had not sieved all the way to $B$. We also, then, replaced the use of the pseudosquares prime test with strong pseudoprime tests [22] using results from [30] so that only a few bases were needed, due to the spotty trial-division information.

(5) We found that, especially for larger $k$, our algorithm spent more time on sieving than prime testing. As mentioned previously, for $k = 3$ the prime testing dominates the running time in practice, and it is worthwhile to use $B = \sqrt{n}$ so that prime testing is not required.

4.1. **Twin primes and Brun's constant.** Let $\pi_2(X)$ count the twin prime pairs $(p, p+2)$ with $p < X$ and let $S_2(X)$ be the sum of the reciprocals of their elements. Thomas Nicely computed these functions up to $2 \cdot 10^{16}$ (see `http://www.trnicely.net/#PI2X`). We verified his computations to 14 digits and extended the results to $X = 10^{17}$. A portion of our computational results are in the table below.

| $X$ | $\pi_2(x)$ | $S_2(X)$ |
|---:|---:|---:|
| $1 \cdot 10^{16}$ | 10304195697298 | 1.8304844246583 |
| $2 \cdot 10^{16}$ | 19831847025792 | 1.8318080634324 |
| $3 \cdot 10^{16}$ | 29096690339843 | 1.8325599218628 |
| $4 \cdot 10^{16}$ | 38196843833352 | 1.8330837014776 |
| $5 \cdot 10^{16}$ | 47177404870103 | 1.8334845790134 |
| $6 \cdot 10^{16}$ | 56064358236032 | 1.8338086822020 |
| $7 \cdot 10^{16}$ | 64874581322443 | 1.8340803303554 |
| $8 \cdot 10^{16}$ | 73619911145552 | 1.8343139034256 |
| $9 \cdot 10^{16}$ | 82309090712061 | 1.8345186031523 |
| $10 \cdot 10^{16}$ | 90948839353159 | 1.8347006694414 |

The last section of Klyve's PhD thesis [18] describes how to use this information to derive bounds for Brun's constant.

We have four remarks on our algorithm implementation:

(1) As mentioned above, for small $k$ like $k = 2$, it is more efficient to set $B = \sqrt{n}$ so that sieving also determines primality, thereby avoiding base-2 strong pseudoprime tests and primality tests.
(2) We computed $S_2$ using Kahan summation [17] with the `long double` data type in C++, which gave us 17 digits, 14 of which were accurate; Thomas Nicely has data with 53 digits of accuracy. The partial sums were accumulated in 10,000 buckets for each process, and then the buckets were in turn added up across processes using Kahan summation.
(3) Our computation took roughly 3 weeks of wall time, which included at least one restart from a checkpoint. Our verification of Nicely's work to $10^{16}$ took 42 hours.
(4) We used a wheel with $W = 6469693230 = 2{\cdot}3{\cdot}5{\cdot}7{\cdot}11{\cdot}13{\cdot}17{\cdot}19{\cdot}23{\cdot}29$. Note that this is roughly $20 \cdot \sqrt{10^{17}}$. There were 214708725 residues $r$ mod $W$ to sieve.

See `OEIS.org` sequence A007508, the number of twin prime pairs below $10^n$.

4.2. **Quadruplet primes.** A related sum involves the reciprocals of the elements of the prime tuple $(p, p+2, p+6, p+8)$. Let $\pi_4(X)$ count these tuplets up to $X$, and let $S_4(X)$ be the sum of the reciprocals of their elements. Thomas Nicely computed these functions up to $2 \cdot 10^{16}$. We extended this computation and partial results are in the table below. The first two lines are Thomas Nicely's own results, which we verified.

| $X$ | $\pi_4(x)$ | $S_4(X)$ |
|---:|---:|---|
| $1 \cdot 10^{16}$ | 25379433651 | 0.8704776912340 |
| $2 \cdot 10^{16}$ | 46998268431 | 0.8704837109481 |
| $3 \cdot 10^{16}$ | 67439513530 | 0.8704870310432 |
| $4 \cdot 10^{16}$ | 87160212807 | 0.8704893020026 |
| $5 \cdot 10^{16}$ | 106365371168 | 0.8704910169467 |
| $6 \cdot 10^{16}$ | 125172360474 | 0.8704923889088 |
| $7 \cdot 10^{16}$ | 143655957845 | 0.8704935288452 |
| $8 \cdot 10^{16}$ | 161868188061 | 0.8704945017556 |
| $9 \cdot 10^{16}$ | 179847459283 | 0.8704953489172 |
| $10 \cdot 10^{16}$ | 197622677481 | 0.8704960981105 |

This computation took about 4 days, and we used a separate program rather than looking for pairs of twin primes in the first program. Even though $k = 4$ is large enough to use prime tests, we found that sieving to $\sqrt{n}$ was faster in practice.

We used a wheel with $W = 200560490130 = 2 \cdot 3 \cdot 5 \cdot 7 \cdot 11 \cdot 13 \cdot 17 \cdot 19 \cdot 23 \cdot 29 \cdot 31$, which gave 472665375 residues.

See `OEIS.org` sequence A050258, the number of prime quadruplets with largest member $< 10^n$.

4.3. **Cunningham chains.** We have two new computational results for Cunningham chains.

(1) We found the smallest chain of length 15 of the first kind, and it begins with the prime

$$p = 90616\ 21195\ 84658\ 42219.$$

The next four chains of this length of the first kind begin with
1 13220 80067 50697 84839
1 13710 75635 40868 11919
1 23068 71734 48294 53339
1 40044 19781 72085 69169
This computation took roughly a month of wall time. Here we used wheel size $W = 19835154277048110 = 2\cdot3\cdot5\cdot7\cdot11\cdot13\cdot17\cdot19\cdot23\cdot29\cdot37\cdot41\cdot43\cdot47$, with 12841500672 residues to sieve. Note that 31 is a rather badly behaved prime for larger Cunningham chains (only 5 residues are excluded), so we left it out of the wheel.

See OEIS.org sequence A005602, smallest prime beginning a Cunningham chain of length $n$ (of the first kind).

(2) In 2008 Jaroslaw Wroblewski found a Cunningham chain of length 17 of the first kind, starting with

$$p = 27\ 59832\ 93417\ 13865\ 93519,$$

and we were able to show that this is in fact the smallest such chain of that length.

This computation took roughly three months of wall time. We used $W = 1051263176683549830 = 2\cdot3\cdot5\cdot7\cdot11\cdot13\cdot17\cdot19\cdot23\cdot29\cdot37\cdot41\cdot43\cdot47\cdot53$, with 35864945424 residues to sieve. With roughly three times as many residues as the previous computation, it took roughly three times as long to complete.

## 5. Discussion and future work

In summary, we have described and analyzed two algorithms for finding primes in patterns and then shown that the second of these algorithms is quite practical by performing a few computations.

We have some ideas for future work.

(1) In the Introduction, we mentioned that our algorithms could be used to find Carmichael numbers by finding prime triplets that satisfy the pattern $(6x + 1, 12x + 1, 18x + 1)$, but we have not yet done that computation [6].
(2) Does it make sense to use Bernstein's doubly focused enumeration to attempt to further reduce the running time? See [5, 29, 31].
(3) A natural extension to our algorithms here is to allow the linear polynomials $f_i$ to potentially be higher degree, irreducible polynomials. See Schinzel's Hypothesis H. (See [27] and [7, §1.2.2]) and the Bateman-Horn conjecture [4].)
(4) An algorithm for twin primes with space roughly $n^{1/3}$ that runs in $O(n/(\log\log n)^2)$ time would be nice.

## Acknowledgments

We wish to thank Frank Levinson for his generous gift that funds the Butler cluster supercomputer Big Dawg and the Holcomb Awards Committee for their financial support of this work.

Thanks to Thomàs Oliveira e Silva for several helpful comments on an earlier version of this paper.

We also wish to thank two anonymous referees for many detailed, helpful comments on earlier versions of this paper. In particular, one of the referees contributed the analysis of the implied constant in §2.3.

## References

[1] M. Agrawal, N. Kayal, and N. Saxena, *PRIMES is in P*, Ann. of Math. (2) **160** (2004), no. 2, 781–793, DOI 10.4007/annals.2004.160.781. MR2123939

[2] A. O. L. Atkin and D. J. Bernstein, *Prime sieves using binary quadratic forms*, Math. Comp. **73** (2004), no. 246, 1023–1030, DOI 10.1090/S0025-5718-03-01501-1. MR2031423

[3] E. Bach and L. Huelsbergen, *Statistical evidence for small generating sets*, Math. Comp. **61** (1993), no. 203, 69–82, DOI 10.2307/2152936. MR1195432

[4] P. T. Bateman and R. A. Horn, *A heuristic asymptotic formula concerning the distribution of prime numbers*, Math. Comp. **16** (1962), 363–367, DOI 10.2307/2004056. MR148632

[5] D. J. Bernstein, *Doubly focused enumeration of locally square polynomial values*, High Primes and Misdemeanours: Lectures in Honour of the 60th Birthday of Hugh Cowie Williams, Fields Inst. Commun., vol. 41, Amer. Math. Soc., Providence, RI, 2004, pp. 69–76. MR2075648

[6] J. Chernick, *On Fermat's simple theorem*, Bull. Amer. Math. Soc. **45** (1939), no. 4, 269–274, DOI 10.1090/S0002-9904-1939-06953-X. MR1563964

[7] R. Crandall and C. Pomerance, *Prime Numbers: A Computational Perspective*, Springer-Verlag, New York, 2001. MR1821158

[8] L. E. Dickson, *A new extension of Dirichlet's theorem on prime numbers*, Messenger of Mathematics **33** (1904), 155–161.

[9] T. Forbes, *Prime clusters and Cunningham chains*, Math. Comp. **68** (1999), no. 228, 1739–1747, DOI 10.1090/S0025-5718-99-01117-5. MR1651752

[10] W. F. Galway, *Dissecting a sieve to cut its need for space*, Algorithmic Number Theory (Leiden, 2000), Lecture Notes in Comput. Sci., vol. 1838, Springer, Berlin, 2000, pp. 297–312, DOI 10.1007/10722028_17. MR1850613

[11] D. M. Gordon and G. Rodemich, *Dense admissible sets*, Algorithmic Number Theory (Portland, OR, 1998), Lecture Notes in Comput. Sci., vol. 1423, Springer, Berlin, 1998, pp. 216–225, DOI 10.1007/BFb0054864. MR1726073

[12] R. K. Guy, *Unsolved Problems in Number Theory*, 3rd ed., Problem Books in Mathematics, Springer-Verlag, New York, 2004. MR2076335

[13] H. Halberstam and H.-E. Richert, *Sieve Methods*, London Mathematical Society Monographs, No. 4, Academic Press [A subsidiary of Harcourt Brace Jovanovich, Publishers], London-New York, 1974. MR0424730

[14] G. H. Hardy and J. E. Littlewood, *Some problems of 'Partitio numerorum'; III: On the expression of a number as a sum of primes*, Acta Math. **44** (1923), no. 1, 1–70, DOI 10.1007/BF02403921. MR1555183

[15] G. H. Hardy and E. M. Wright, *An Introduction to the Theory of Numbers*, 5th ed., The Clarendon Press, Oxford University Press, New York, 1979. MR568909

[16] H. A. Helfgott, *An improved sieve of Eratosthenes*, Math. Comp. **89** (2020), no. 321, 333–350, DOI 10.1090/mcom/3438. MR4011545

[17] W. Kahan, *Pracniques: Further remarks on reducing truncation errors*, Commun. ACM **8** (1965), no. 1, 40ff.

[18] D. Klyve, *Explicit bounds on twin primes and Brun's constant*, ProQuest LLC, Ann Arbor, MI. Thesis (Ph.D.)–Dartmouth College, 2007. MR2712414

[19] H. W. Lenstra Jr. and C. Pomerance, *Primality testing with Gaussian periods*, J. Eur. Math. Soc. (JEMS) **21** (2019), no. 4, 1229–1269, DOI 10.4171/JEMS/861. MR3941463

[20] G. Löh, *Long chains of nearly doubled primes*, Math. Comp. **53** (1989), no. 188, 751–759, DOI 10.2307/2008735. MR979939

[21] R. F. Lukes, C. D. Patterson, and H. C. Williams, *Some results on pseudosquares*, Math. Comp. **65** (1996), no. 213, 361–372, S25–S27, DOI 10.1090/S0025-5718-96-00678-3. MR1322892

[22] G. L. Miller, *Riemann's hypothesis and tests for primality*, J. Comput. System Sci. **13** (1976), no. 3, 300–317, DOI 10.1016/S0022-0000(76)80043-8. MR480295

[23] T. R. Nicely, *Enumeration to* $10^{14}$ *of the twin primes and Brun's constant*, Virginia J. Sci. **46** (1995), no. 3, 195–204. MR1401560

[24] C. Pomerance, *On the distribution of pseudoprimes*, Math. Comp. **37** (1981), no. 156, 587–593, DOI 10.2307/2007448. MR628717

[25] H. Riesel, *Prime Numbers and Computer Methods for Factorization*, 2nd ed., Progress in Mathematics, vol. 126, Birkhäuser Boston, Inc., Boston, MA, 1994. MR1292250

[26] J. B. Rosser and L. Schoenfeld, *Approximate formulas for some functions of prime numbers*, Illinois J. Math. **6** (1962), 64–94. MR0137689

[27] A. Schinzel and W. Sierpiński, *Sur certaines hypothèses concernant les nombres premiers* (French), Acta Arith. **4** (1958), 185–208; erratum **5** (1958), 259, DOI 10.4064/aa-4-3-185-208. MR106202

[28] J. P. Sorenson, *The pseudosquares prime sieve*, Algorithmic Number Theory, Lecture Notes in Comput. Sci., vol. 4076, Springer, Berlin, 2006, pp. 193–207, DOI 10.1007/11792086_15. MR2282925

[29] J. P. Sorenson, *Sieving for pseudosquares and pseudocubes in parallel using doubly-focused enumeration and wheel datastructures*, Algorithmic Number Theory, Lecture Notes in Comput. Sci., vol. 6197, Springer, Berlin, 2010, pp. 331–339, DOI 10.1007/978-3-642-14518-6_26. MR2721430

[30] J. Sorenson and J. Webster, *Strong pseudoprimes to twelve prime bases*, Math. Comp. **86** (2017), no. 304, 985–1003, DOI 10.1090/mcom/3134. MR3584557

[31] K. Wooding and H. C. Williams, *Doubly-focused enumeration of pseudosquares and pseudocubes*, Algorithmic Number Theory, Lecture Notes in Comput. Sci., vol. 4076, Springer, Berlin, 2006, pp. 208–221, DOI 10.1007/11792086_16. MR2282926

[32] T. Z. Xuan, *Integers free of small prime factors in arithmetic progressions*, Nagoya Math. J. **157** (2000), 103–127, DOI 10.1017/S0027763000007212. MR1752478

[33] Y. Zhang, *Bounded gaps between primes*, Ann. of Math. (2) **179** (2014), no. 3, 1121–1174, DOI 10.4007/annals.2014.179.3.7. MR3171761

Department of Computer Science and Software Engineering, Butler University, Indianapolis, Indiana 46208

*Email address*: sorenson@butler.edu

Department of Mathematics, Statistics, and Actuarial Science, Butler University, Indianapolis, Indiana 46208

*Email address*: jewebste@butler.edu