Improved Incremental Prime Number Sieves

Paul Pritchard

Griffith University, School of Computing and Information Technology, Queensland, Australia 4111

Abstract. An algorithm due to Bengalloun that continuously enumerates the primes is adapted to give the first prime number sieve that is simultaneously sublinear, additive, and smoothly incremental:

- it employs only $\Theta(n/\log\log n)$ additions of numbers of size $O(n)$ to enumerate the primes up to n, equalling the performance of the fastest known algorithms for fixed n;
- the transition from n to $n+1$ takes only $O(1)$ additions of numbers of size $O(n)$. (On average, of course, $O(1)$ such additions increase the limit up to which all primes are known from n to $n+\Theta(\log\log n)$).

1 Introduction

A so-called "formula" for the i'th prime has been a long-lived concern, if not quite the Holy Grail, of Elementary Number Theory. This concern seems poorly motivated, as evidenced by the extraordinary freak-show of solutions proffered over the ages. The natural setting is Algorithmic Number Theory, and what is desired is much better cast as an algorithm to compute the i'th prime. Given that approaches involving (all) smaller primes have been deemed acceptable, the problem can perhaps best be formulated as that of finding an algorithm to enumerate the primes, with efficiency and (as we shall see) incrementality the desirable properties.

The problem of enumerating the prime numbers is one of the most venerable of algorithmic problems. It received a deceptively simple and efficient solution from Eratosthenes of Alexandria in the 3rd century B.C. His insight was to recast the problem as that of removing the non-primes up to a fixed limit N. Eratosthenes' sieve requires $\Theta(N\log\log N)$ additions of numbers of size $O(N)$.

To paraphrase a remark of C.A.R. Hoare about Algol 60, Eratosthenes' sieve was an improvement on most of its successors. In this connection, note that the asymptotically fastest known multiplication algorithm for a RAM, that of Schönhage and Strassen (see [2]), has a complexity equal to that of $\Theta(\log\log N\log\log\log N)$ additions. So the bit-complexity of Eratosthenes' sieve is lower than one of $\Theta(N)$ multiplications. The latter is characteristic of many proposed *parallel* algorithms (let alone some sequential ones).

Nevertheless, progress was eventually made. The fastest known (sequential) algorithm is now our wheel sieve [9, 10], which requires $\Theta(N/\log\log N)$ additions. It enjoys the properties of being *sublinear*, i.e., $o(N)$, and *additive*, i.e., not requiring multiplications.

Bengalloun [3] promoted another desideratum, that of being *incremental*. This means that N need not be fixed, so that the algorithm can in theory be run indefinitely to enumerate the primes. Bengalloun's incremental sieve requires $\Theta(N)$ multiplications (and additions) to find the primes up to N. Any sieve algorithm can be made incremental by repeatedly running it with a doubled limit. But Bengalloun's is incremental in a much more natural way: it takes bounded time to progress from N to $N+1$, for all N. We shall describe it as *smoothly incremental*.

Bengalloun also outlined how our technique of using wheels could be incorporated in his basic algorithm to reduce its complexity to $\Theta(N/\log\log N)$ multiplications, but observed that our method in [9] of avoiding multiplications would require a prohibitively large number of multiplication tables.

The main contribution of this paper is to exhibit an algorithm that simultaneously enjoys all these desirable properties: it is a sublinear, additive, and smoothly incremental prime number sieve. The construction of this algorithm proceeds in two phases.

The first phase modifies Bengalloun's linear (i.e., $\Theta(N)$) smoothly incremental but multiplicative sieve to avoid multiplications. It does so by calculating differences using an incrementally grown multiplication table. Theorems by Heath-Brown and Iwaniec show that the table does not grow too quickly.

The second phase shows how dynamic wheels, i.e., reduced residue systems of the product of the primes up to a limit that grows with N, can be used to reduce the complexity while still avoiding multiplications and retaining the property of being smoothly incremental.

2 Notation

We employ a consistent notation, due to E. W. Dijkstra, wherever a variable is bound by a quantifier such as $\forall$, $\exists$, $\sum$, $\prod$, **Max**, **Min** and $\{\}$. The set-forming quantifier $\{\}$ is our invention; it was first used in [13]. The general form for a set-constructor is $(\{\}x : D(x) : t(x))$ which denotes the set of values $t(x)$ when x ranges over the domain characterized by $D(x)$. However, $t(x)$ is commonly x, and in such cases we omit the second colon and term, giving a form very close to traditional mathematical notation. Such an abbreviation is also used with $\sum$, $\prod$, **Max** and **Min**.

The notion of a *wheel* is central to the algorithms to be discussed. The k'th wheel W_k ($k > 0$) is a particular reduced residue system of the product of the first k primes. In more prosaic terms, W_k is the set of all natural numbers no greater than the product of the first k primes and not divisible by one of the first k primes. W_k^* is the set of all natural numbers not divisible by one of the first k primes. g_k is the maximum gap between successive natural numbers not divisible by one of the first k primes. We use the following notation (introduced in [11]):

$\emptyset$: the empty set;
$|S|$: the cardinality of set S;
$next(S, x)$: $(Min\ y : y \in S \wedge y > x)$;
$a..b$: $(\{\}x : a \leq x \leq b)$;
(x, y): the g.c.d. of x and y;
$x \mid y$: x divides y;
p_i: the i'th prime number;
$Primes(S)$: $(\{\}x : x \in S \wedge x$ is a prime number$)$;
$\pi(n)$: $|Primes(2..n)|$;
d_n: $(Max\ i : p_i \leq n : p_i - p_{i-1})$, $n > 2$;
Π_k: $(\prod i : 1 \leq i \leq k : p_i)$, with $\Pi_0 = 1$;
W_k: $(\{\}x : 1 \leq x \leq \Pi_k \wedge (x, \Pi_k) = 1)$;
$W_k^{(i)}$: the i'th greatest member of W_k;
W_k^* : $(\{\}x : 1 \leq x \wedge x\ mod\ \Pi_k \in W_k)$;
g_k: $(Max\ x : x \in W_k^* : next(W_k^*, x) - x)$.

Our algorithms are expressed in the language of guarded commands used in [4], extended with **if**- and **forall**-commands. The command

$$if\ b\ then\ S\ fi$$

is an abbreviation of

$$if\ b \rightarrow S\ [\!]\ \neg b \rightarrow skip\ fi.$$

The **forall**-command denotes iteration over a fixed finite set in unspecified order (see [12]).

3 Bengalloun's Sieve

The following discussion of Bengalloun's sieve recapitulates our presentation in [12].

Bengalloun's basic sieve is based on the following normal form for composites c:

$$c = p \cdot f \text{ where } p = lpf(c) \text{ and } f > 1 \tag{1}$$

"lpf" denotes the least prime factor function. Note that in (1) we have $lpf(f) \geq p$ and $lpf(c) \leq \sqrt{c}$.

Bengalloun's sieve tabulates the function lpf on a superset of an incrementally increasing segment $2..n$ of the natural numbers. $lpf(n)$ is tabulated when n is processed if n is even, otherwise it is tabulated when processing the largest composite $< n$ with the same value of f in its normal form. Changing perspective, when composite $n = p_i \cdot f$ is processed, lpf is tabulated at $p_{i+1} \cdot f$, provided $p_i < lpf(f)$. The primes are gathered in an array as they are discovered.

In our presentation of the algorithm, $f|_S$ denotes the restriction of the function f to the sub-domain S.

Bengalloun's Sieve:

```
$n, P, LPF := 2, \{2\}, \{(2,2)\}$;
$\{P = Primes(2..n) \wedge LPF = lpf|_{domain(LPF)} \wedge domain(LPF) = (2..n) \cup$
 $(\{\}p, f : f > 1 \wedge p \in Primes(3..lpf(f)) \wedge prev(P,p) \cdot f \leq n : p \cdot f)\}$
$do\ true \rightarrow$
   $n := n + 1$;
   $if\ 2 \mid n \rightarrow LPF := LPF \cup \{(n,2)\}$
   $[\!]\ \neg(2 \mid n) \rightarrow skip$
   $fi$;
   $if\ \neg(n \in domain(LPF)) \rightarrow P, LPF := P \cup \{n\}, LPF \cup \{(n,n)\}$
   $[\!]\ n \in domain(LPF) \rightarrow$
     $p := LPF(n)$;
     $f := n \div p$;
     $if\ p < LPF(f) \rightarrow p' := next(P,p)$;
                         $LPF := LPF \cup \{(p' \cdot f, p')\}$
     $[\!]\ p = LPF(f) \rightarrow skip$
     $fi$
   $fi$
$od$
```

A neat implementation uses a single array lpf with the representation invariant

$$lastp = max(P) \wedge lpf[x] = \begin{cases} 0, & \text{if } 2 \leq x \leq 2n \wedge \\ & \neg(x \in domain(LPF)), \\ LPF(x), & \text{if } x \text{ is composite} \wedge \\ & x \in domain(LPF), \\ next(P,x), & \text{if } x \in Primes(2..(lastp-1)). \end{cases}$$

Bertrand's theorem — there is always a prime between n and $2n$ — justifies only defining array lpf up to index $2n$.

The algorithm clearly requires $\Theta(N)$ multiplications and additions to reach $n = N$.

4 Avoiding Multiplications in Bengalloun's Sieve

Multiplicative operations occur in two places in Bengalloun's sieve. The first is the computation $f := n \div p$ after p has been looked-up in $lpf[n]$. This is avoided by the simple expedient of also tabulating the co-factor f along with the least prime factor. Another array, cf (for co-factor) is used for this purpose. So the multiplicative operation is replaced with $f := cf[n]$.

The other multiplicative operation occurs in the abstract operation

$$LPF := LPF \cup \{(p' \cdot f, p')\} \qquad (2)$$

which is now implemented as

```
$n' := p' \cdot f$;
$lpf[n'], cf[n'] := p', f$
```

Since $p' \cdot f = p \cdot f + (p' - p) \cdot f$, and $p \cdot f = n$, the problem of computing $p' \cdot f$ reduces to that of computing $(p' - p) \cdot f$. Suppose, after it is computed, this latter product and the value f are recorded abstractly as $\Delta(p)$ and $lastf(p)$ respectively. (We shall postpone implementation details.)

Now when a value n is processed, and $p' \cdot f$ must be computed, the product may be written as

$$n + (p' - p) \cdot lastf(p) + (p' - p) \cdot (f - lastf(p)), \tag{3}$$

provided $lastf(p)$ is defined, in which case the second summand is $\Delta(p)$, which may be looked up, and the third involves very small multiplicands.

Under this provisional arrangement $lastf(p_i)$ and $\Delta(p_i)$ are initialised when processing $n = p_i \cdot p_{i+1}$, so only in this case is $lastf(p_i)$ undefined. Then $p' \cdot f$ is p_{i+1}^2. It is possible to compute this value incrementally without using multiplications, but we shall adopt a more straightforward approach.

When $p = f$, so that $n = p^2$, $lastf(p)$ is initialised to p and $\Delta(p)$ to $(p' - p) \cdot p$. The effect is to extend the invariant with the conjunct

$$(\forall i : p_i^2 \leq n : lastf(p_i) = (Max\ f : f = p_i\ \vee (lpf(f) > p_i \wedge p_i \cdot f \leq n)) \wedge$$
$$\Delta(p_i) = (p_{i+1} - p_i) \cdot lastf(p_i))$$

Now the product $p' \cdot f$ may always be written as (3) above.

We shall arrange for a multiplication table to be incrementally tabulated (together with row-offsets) so that the third summand $(p' - p) \cdot (f - lastf(p))$ may be looked-up (without a multiplication).

The first multiplicand is $p' - p$. Since $n = p \cdot f$ and $f \geq p'$, $n \geq p \cdot p'$, so $p < \sqrt{n}$ and hence $p' < 2\sqrt{n}$ by Bertrand's Theorem. Therefore

$$p' - p < d_{2\sqrt{n}}.$$

The best known upper bound on prime gaps is the very conservative

$$d_n = O(n^{0.55+\epsilon}) \text{ for any } \epsilon > 0$$

[6]. Thus

$$p' - p = O(n^{0.275+\epsilon}) \text{ for any } \epsilon > 0.$$

Now consider the second multiplicand. $lastf(p_i)$ and f are either p_i and p_{i+1} respectively, or successive members of W_{i+1}^*. In the former case,

$$f - lastf(p_i) = O(n^{0.275+\epsilon})$$

as above. In the latter,

$$f - lastf(p_i) = O(g_{i+1}) = O(p_{i+1}^2)$$

by a result of Iwaniec [7]. Since $f = n \div p_i$, the difference is also bounded by

$$d_{n \div p_i} = O((n \div p_i)^{0.55+\epsilon}).$$

We may approximately balance these two known bounds as follows. If $p_i = O(n^{0.216})$, the difference is $O(p_{i+1}^2) = O(n^{0.432})$. Otherwise, $p_i = \Omega(n^{0.216})$, and the difference is $O((n \div p_i)^{0.55+\epsilon}) = O(n^{0.432})$.

We are thus able to prove that an $O(n^{0.275+\epsilon})$ by $O(n^{0.432})$ multiplication table suffices for the product $(p' - p) \cdot (f - \mathit{lastf}(p))$.

In passing, we note that this is certainly a gross over-estimate. According to [1], Cramér's conjecture that $d_n = \Theta(\log^2 n)$ "has been called into question", but the slightly weaker

$$d_n = O(\log^{2+\epsilon} n) \text{ for any } \epsilon > 0$$

is "still probably true". Suppose this latter claim is true. Then when $p_i = O(\log n)$, the difference is

$$O(p_{i+1}^2) = O(\log^2 n).$$

And when $p_i = \Omega(\log n)$, the difference is

$$O(\log^{2+\epsilon}(n \div p_i)) = O(\log^{2+\epsilon} n).$$

So it is probably the case that a square multiplication table of side $O(\log^{2+\epsilon} n)$ suffices!

For simplicity we construct a square multiplication table. Because of symmetry, we only construct the upper triangle, going down each column in order. A one-dimensional array is used, so that $i \cdot j$ is stored at index $j(j-1)/2 + i$, for $j \geq i$. In order to avoid a multiplication when accessing the table, the values j^2 are incrementally computed and stored as well. With this scheme, a full m by m table is available after $m(m+1)/2$ entries have been made. c_1 entries are added at each iteration (i.e., for each value of n), for an appropriate constant c_1, which is guaranteed to exist because each side is $O(\sqrt{n})$. It is very likely that $c_1 = 1$ suffices. Each entry may be computed in $O(1)$ additions.

It remains to implement the abstract functions Δ and lastf. These are merely tabulated in arrays, indexed either by their prime argument (which is wasteful), or its index (in which case the prime indices of the least prime factors would need to be stored in array lpf). Because a bounded number of one-dimensional arrays is used, they may be interleaved in a single incrementally growing one-dimensional array with $O(n)$ elements, without requiring multiplications for access.

The resulting algorithm is not only incremental, linear and additive, but it is optimally smooth at the bit-complexity level — each value of n is processed in $O(1)$ additions of numbers of size $O(n)$. Note that performing just one multiplication would vitiate the latter property.

5 Using Wheels to Add Sublinearity

As was observed by its inventor in [3], Bengalloun's Sieve may be sped up by exploiting the same simple idea powering our wheel sieve (see [10]): a number n exceeding Π_k can only be prime if $n \in W_k^*$, for otherwise it is divisible by one of the first k primes.

To derive maximum benefit, k is maximised. In order to do so, the wheel that is maintained must be updated to W_{k+1} when n passes Π_{k+1}. This is accomplished by incrementally building W_{k+1} while the complete wheel W_k is used to generate candidates n.

So rather than simply incrementing n, our new algorithm updates n to $next(W_k^*, n)$, where k is maximal such that $\Pi_k < n$. Since n will be odd, the first **if**-command is no longer needed.

Now suppose composite $n = p \cdot f$ is processed by this modified algorithm, so $p \geq p_{k+1}$. If $p < lpf(f)$, then $p' \cdot f \in W_{k+1}^*$, so it too will later be processed. The present code caters for this by tabulating lpf at $p' \cdot f$. The only concern, therefore, is that the least composite number n with a given co-factor f that is processed by the new algorithm is factored.

These numbers are just those of the form

$$p_{k+1} \cdot f \text{ where } k \geq 1 \text{ and } f \in W_k - W_{k-1}.$$

The first of them, 25, is treated specially, and the factorisation of each of the others is recorded when the previous one n in numerical order is processed. Let $n = p_{k+1} \cdot f$. There are two cases, depending on whether f is the greatest member of W_k.

If not, then with f' the succeeding member,

$$p_{k+1} \cdot f' = n + p_{k+1} \cdot (f' - f). \tag{4}$$

Consider the product in (4). The multiplicand $f' - f$ falls within the bounds of the multiplication table. To bound the other multiplicand, note that

$$\Pi_k < n < \Pi_{k+1}$$

by definition of k. Since

$$\sum_{p \leq x} \log p \sim x$$

(see [5, theorems 420 and 434]),

$$p_{k+1} = \Theta(\log n).$$

Hence the product may be looked up in the multiplication table.

In the case that f is the greatest member $\Pi_k - 1$, the next number is $p_{k+2} \cdot (\Pi_k + 1)$, which differs from n by

$$(p_{k+2} - p_{k+1}) \cdot \Pi_k + p_{k+2} + p_{k+1}. \tag{5}$$

Unfortunately, Π_k falls outside the bounds of the multiplication table. However, the product in (5) may be incrementally computed by repeated additions of Π_k. Since

$$p_{k+2} - p_{k+1} = O(d_{p_{k+2}}) = \Theta(\log^{0.55+\epsilon} n),$$

there is ample time to do the calculation after n reaches Π_k.

The incremental construction of W_{k+1} is straightforward. The numbers $n > \Pi_k$ with $lpf(n) > p_{k+1}$ may simply be linked as encountered, and those less than Π_k linked in when used for the last time. The otherwise unused even-numbered elements of array lpf may be used to construct the linked list. Alternatively, two new arrays may be used: one to hold the current wheel W_k, while the next wheel W_{k+1} is built in the other.

Both new incremental sub-computations — building the next wheel and the product of Π_k — may be carried out in $O(1)$ additions of numbers of size $O(n)$ for each of the $O(n/\log\log n)$ numbers up to n that are processed.

We have arrived at an algorithm that

- employs only $\Theta(n/\log\log n)$ additions of numbers of size $O(n)$ to enumerate the primes up to n, equalling the performance of the fastest known algorithms for fixed n;
- moves from n to $n+1$ in only $O(1)$ additions of numbers of size $O(n)$. (On average, of course, $O(1)$ such additions increase the limit up to which all primes are known from n to $n + \Theta(\log\log n)$).

A remark on the machine model is called for. We have been implicitly assuming a RAM with a word size that grows with the computation, so that the running time to find all primes up to n is $O(n/\log\log n)$ arithmetic operations on numbers of size $O(n)$. The reader may be more comfortable with a model positing a fixed size word, in which case the numbers used may be represented by linked lists of the appropriate length, with no change to the underlying bit-complexity.

6 Closing Remarks

Since the author's discovery of the *wheel sieve* in 1979, no algorithm for finding the primes up to even fixed N has been discovered with a complexity of $o(N/\log\log N)$ arithmetic operations. We conjecture that $O(N/\log\log N)$ additions of numbers of size $O(N)$ is best possible. We have achieved this with a smoothly incremental algorithm.

Determining the status of the above conjecture, or more generally, giving good lower bounds on the time-complexity of the problem of enumerating the primes up to N (whether or not by incremental algorithms), are daunting open problems. $\Omega(N/\log N)$ additive operations on numbers of size $O(N)$ are needed to list the primes up to N. The same lower bound of $\Omega(N)$ bit operations applies if output as a bit-vector is permitted. But if output in the form of prime gaps is allowed (and why not?), even this lower bound has yet to be established, since we showed in [11] that the primes up to N may be stored in $O(N \log\log N/\log N)$ bits and still recovered in order in $O(1)$ additive operations per prime.

Our new algorithm lacks one important property: it is not *compact*. As we showed in [8, 11], Eratosthenes' sieve and some linear wheel-based sieves can be implemented to run in $O(N^{0.5})$ bits, whereas our new algorithm and Bengalloun's sieve require $O(N \log N)$ bits (to enumerate the primes up to N).

The space requirement of our new algorithm may be reduced by a factor of $1/\Theta(\log\log n)$, while preserving its sublinear, additive and smoothly incremental properties, by adapting the invertible mapping technique introduced in section 7 of [9], but the other techniques presented therein rely on a bounded problem size, and do not readily adapt to an incremental setting.

Little is known about lower bounds for space for sublinear algorithms for finding the primes, whether or not the algorithms are incremental. The known upper bounds exhibit a great disparity in moving from a linear to a sublinear algorithm. Is there a sublinear algorithm with a space complexity of $o(N^{1-\epsilon})$ bits for some $\epsilon > 0$?

As has been mentioned, great reductions in space requirements for non-incremental algorithms may be obtained by trading in sublinearity. We plan to investigate the extent to which this is possible for incremental algorithms in a future paper.

References

1. Adleman, L.M. and McCurley, K.S.: Open problems in number theoretic complexity, II. Proceedings of the First Algorithmic Number Theory Symposium. This volume.
2. Aho, A., Hopcroft, J., Ullman., J.: The Design and Analysis of Computer Algorithms. Addison-Wesley, Reading, Massachusetts, 1974.
3. Bengalloun, S.: An incremental primal sieve. Acta Informatica **23** (1986) 119–125
4. Gries, D.: The Science of Programming. Springer-Verlag, New York, 1981.
5. Hardy, G.H. and Wright, E.M.: An Introduction to the Theory of Numbers, 5th ed. London Univ. Press, London, 1979.
6. Heath-Brown, D.R., Iwaniec, H.: On the difference between consecutive primes. Inventiones Mathematicae **55** (1979) 49–69
7. Iwaniec, H.: On the problem of Jacobsthal. Demonstratio Math. **11** (1978) 225–231
8. Pritchard, P.: On the prime example of programming. In Tobias, J. (ed.): Language Design and Programming Methodology. Lecture Notes in Computer Science **79**, (1980) 85–94
9. Pritchard, P.: A sublinear additive sieve for finding prime numbers. Comm. ACM **24** (1981) 18–23
10. Pritchard, P.: Explaining the wheel sieve. Acta Informatica **17** (1982) 477–485
11. Pritchard, P.: Fast compact prime number sieves (among others). J. Algorithms **4** (1983) 332–344.
12. Pritchard, P.: Linear prime number sieves: a family tree. Sci. Comp. Prog. **9** (1987) 17–35
13. Pritchard, P.: Opportunistic algorithms for eliminating supersets. Acta Informatica **28** (1991) 733–754