FAST COMPACT PRIME NUMBER SIEVES
(AMONG OTHERS)

Paul Pritchard

TR 81-473
October 1981

Department of Computer Science
Cornell University
Ithaca, New York 14853

1. Introduction

The problem of finding all prime numbers up to a given limit N has received extensive attention from computer scientists (as well as from the librarian at Ptolemy's library at Alexandria in the third century B.C.!). Dijkstra [4], Gries and Misra [6] and Pritchard [17] have tackled the problem as an exercise in programming methodology; Mairson [14] studied its algorithmic complexity; Bays and Hudson [2] and Brent [3] presented practical computational techniques for its solution for large N.

Recently, Pritchard [18] discovered the first sublinear solution, i.e. one that requires only o(N) arithmetic operations[1] (and additions, at that). However, it uses too much storage to compete in practical applications with versions of the librarian Eratosthenes' solution, which are compact in the sense of using only $O(\sqrt{N})$ bits of storage [2,17]. It is natural to ask whether there are faster compact algorithms, and [18] answered "yes" by promising "a new practical algorithm, which runs in $\Theta(\sqrt{N})$ bits with an arithmetic complexity of $\Theta(N)$ additions".

This paper delivers the promised algorithm, and a little more. First, a parameterized family of algorithms is presented which includes Eratosthenes' classical algorithm and several others [2,4,11,12]. Then it is shown how a particular choice of parameters leads to a compact, linear, additive algorithm, and finally that the latter's storage requirement can be reduced to $o(\sqrt{N})$ bits. This last optimization utilizes a provably compact method of storing all the primes up to some limit.

[1]Our notation for functional relationships is given under "Notation and Definitions" below.

2. Presenting the Family

In [19], the sublinear sieve (as we shall refer to it) is presented in terms of <u>wheels</u>. The k'th wheel W_k is a particular reduced residue class (see [13]) of the product of the first k primes.

<u>Notation and Definitions</u>

$\emptyset$: the empty set;

$|S|$: the cardinality of set S;

next(S,x) : $\min\{y \mid y \in S$ **and** $y > x\}$;

a..b : $\{x \mid a \le x \le b\}$;

(x,y) : the greatest common divisor of x and y;

$x|y$: x divides y;

p_i : the i'th prime number;

Primes(S) : $\{x \mid x \in S$ **and** x prime$\}$;

$\pi(n)$: |Primes(2..n)|;

d_n : $\max\{p_i - p_{i-1} \mid p_i \le n\}$;

Π_k : $\prod_{i=1}^{k} p_i$, $(\Pi_0 = 1)$;

W_k : $\{x \mid 1 \le x \le \Pi_k$ **and** $(x, \Pi_k) = 1\}$ $(k \ge 0)$ -- the k'th wheel;

$W_k^{(i)}$: the i'th greatest member of W_k;

W_k^* : $\{x \mid 1 \le x$ **and** x **mod** $\Pi_k \in W_k\}$;

g_k : $\max\{\text{next}(W_k^*, x) - x \mid x \in W_k^*\}$ -- the maximum gap on the k'th wheel.

Our notation for various functional relationships is standard:

$O(f(n))$: order at most f(n);

$\Omega(f(n))$: order at least f(n);

$\Theta(f(n))$: order exactly f(n);

$o(f(n))$: order less than $f(n)$;

$\sim f(n)$: asymptotic to $f(n)$.

W_k^*, said to be the result of "rolling" W_k, is the set of all positive numbers not divisible by one of the first k primes. W_0^* is the set of all positive numbers.

The sublinear sieve, in the abstract, consists of a single loop which keeps invariant $W = W_k \cap 1..N$ **and** $p = p_{k+1}$ while increasing k until $p^2 > N$. Thus the loop terminates with $W = W_{\pi(\sqrt{N})} \cap 1..N$, and the primes can be found because

$$\text{Primes}(2..N) = (W^*_{\pi(\sqrt{N})} \cap 2..N) \cup \text{Primes}(2..\sqrt{N}) \tag{1}$$

The sublinear sieve is a _dynamic_ wheel sieve, meaning the wheel changes in that at each iteration a new wheel W_k is used. In contrast, the following family of algorithms uses a _fixed_ wheel W_k where k is a parameter of the family. The mathematical motivation is that

$$W_h^* = W_k^* - \{p_i \cdot f \mid f \in W_k^* \textbf{ and } k < i \le h\}, \quad (0 \le k \le h) \tag{2}$$

Choosing $h = \pi(\sqrt{N})$ in (2), and substituting in (1), leads to

$$\text{Primes}(2..N) = (W_k^* \cap 2..N) - \{p \cdot f \mid f \in W_k^* \textbf{ and } p \in \text{Primes}(p_{k+1}..\sqrt{N})\} \cup \text{Primes}(2..\sqrt{N}) \tag{3}$$

(3) shows quite explicitly how $\text{Primes}(1+\sqrt{N}..N)$ is determined by $\text{Primes}(p_{k+1}..\sqrt{N})$ and W_k. Our family of algorithms is directly based on (3).

Compact versions of Eratosthenes' sieve [2,3,17] are obtained by operating on successive segments $a_m..b_m$ of $1..N$, where $b_m = m \cdot \Delta$ and $a_m = b_m - \Delta + 1$, for some integer constant $\Delta>0$. Our family also uses this technique, with Δ as the second parameter. Each member is thus a <u>segmented</u> <u>fixed-wheel</u> <u>sieve</u> (SFWS).

In order to avoid unimportant complications, the (parameterized) algorithm uses operations on the sets $\mathrm{Primes}(p_{k+1}..\sqrt{N})$ and W_k^*. We deal later with the implementation of these operations. The task of the algorithm is to compute $\mathrm{Primes}(1+\sqrt{N}..N)$. Our presentation uses a history variable P to accumulate the primes. Storage for set P is not counted, as in practice the primes would be printed or sent to some external storage device. We assume $\Delta|N$ and $k \le \pi(\sqrt{N})$ since the assertions need to be adjusted slightly otherwise. Our algorithms are presented in guarded command notation [5].

```
SFWS(k,Δ):
  {$N > 0,\ \Delta > 0,\ 0 \le k \le np = \pi(\sqrt{N}),\ \Delta|N$}
  initialise;
  {**invariant**: $a=(m-1)\cdot\Delta+1,\ b = m\cdot\Delta,\ P = \mathrm{Primes}(1+\sqrt{N}..a-1)$}
  **do** $a \le N$ → S:= a..b;
              sift S;
              {$W_k^* \cap S = W_{np}^* \cap a..b$}
              "accumulate the primes in S":
                 $P := P \cup (W_k^* \cap S - \{1\})$;
              a,b,m:= b+1, b+Δ, m+1
  **od**
  {$P = \mathrm{Primes}(1+\sqrt{N}..N)$}
```

In order to do the sifting efficiently, it is helpful to introduce

var factor: **array**$[k+1..\pi(\sqrt{N})]$ **of** $1..\Pi_k$

and to extend the main invariant above with the following clause:

$$(\forall i{:}k<i\leq np{:}\ \text{factor}[i] = \min\{f\in W_k^* \mid p_i\cdot f \geq a\}) \tag{4}$$

(4) states that the factor f of the next multiple $p_i\cdot f$ of p_i to be deleted from S is in factor[i]. Our refinement uses this information to implement an induction on the h of (2).

```
"sift S":
   i,p:= k+1,$p_k$;
   {invariant: $W_k^* \cap S = W_{i-1}^* \cap a..b$}
   do i ≤ np →
     "sift with $p_i$":
        p,f:= next(Primes($p_{k+1}..\sqrt{N}$),p), factor[i];
        mult:= p·f;
        do mult ≤ b  →  S,f:= S - {mult}, next($W_k^*$,f);
                        mult:= p·f
        od;
     factor[i],i:= f,i+1
   od
```

In order to show how the primes that are left in a..b after sifting may be explicitly found, we give a lower-level refinement. Here it is helpful to extend our invariant with

$$\text{nexty} = \text{next}(W_k^*-\{1\},a-1) \tag{5}$$

(5) means that the next member of a..b to be examined for primality is nexty. This and (1) lead to:

```
"accumulate the primes in S":
   "P:= P ∪ ($W_k^*$ ∩ S - {1})":
      {P = $P_0$}
      y:= nexty;
      {**invariant**: P = $P_0$ ∪ ($W_k^*$ ∩ a..y-1 - {1})}
      **do** y ≤ b → **if** y ∈ S → P:= P ∪ {y} [] y ∉ S → skip **fi**;
                    y:= next($W_k^*$,y)
      **od**;
      nexty:= y
```

Finally, the (now extended) main invariant must be established:

```
   initialise:
      a,b,m,P:= 1,Δ,1,∅;
      i:= k+1; **do** i ≤ np → factor[i]:= 1; i:= i+1 **od**;
      nexty:= $p_{k+1}$
```

The correctness of this family of abstract algorithms is apparent from the invariants and equations (1-5). Minor optimizations are possible; for instance, the initial segment a..b could be taken as $1+\sqrt{N}..\Delta+\sqrt{N}$; also, it is not necessary to sift with p_i if $p_i^2 > b$.

Particular instantiations of the parameters k,Δ give the abstract algorithms underlying several different solutions:

(i) (Δ,k) = (N,0): Eratosthenes' sieve.

(ii) (Δ,k) = (N,1): A common variation of Eratosthenes' sieve that avoids the even numbers (e.g. [11, p.394]).

(iii) $(\Delta,k) = (N,2)$: A version of Eratosthenes' sieve that avoids multiples of 2 and 3. The version in [12, p.617] does this, but it is implemented with a priority queue and has inferior time and space complexities.

(iv) $(\Delta,k) = (\sqrt{N},0)$: The segmented sieve of Eratosthenes [2,17]. $\Delta = \Omega(\sqrt{N})$ is needed in order not to degrade the arithmetic complexity (but see §4); SFWS$(\theta(\sqrt{N}),c)$, for any constant $c \geq 0$, has the same time/space complexities (up to constant factors).

(v) $(\Delta,k) = (2,1)$: Dijkstra's algorithm [4] (after transforming the loop for "sift S" so that it terminates if b is found to be nonprime).

(vi) $(\Delta,k) = (\sqrt{N}, \max\{k \mid \Pi_k \leq \sqrt{N}\})$: A compact linear sieve; its space/time complexity is analysed in §3.

(vii) $(\Delta,k) = (N, \max\{k \mid \Pi_k \leq N\})$: Interestingly, this does <u>not</u> give the sublinear sieve, but only a linear sieve. The sublinear sieve obtains its extra efficiency by merging S and W_k in a single dynamic data structure.

A large wheel has been employed in a previous sieve algorithm -- Brent [3] used W_7 in a clever way (see [19]) to speed up a segmented sieve when sifting with $p_1,\ldots,p_7$. Although useful in practice, this technique does not improve on the asymptotic complexity of Eratosthenes' sieve and, in any case, the effect is very simply obtained by our code for "accumulate the primes in S".

3. An Implementation and its Complexity

In implementing SFWS, we try to minimize the storage requirement and to avoid multiplications. The details are as follows. S always represents a subset of a..b. So we declare

```
var S: array[0..Δ-1] of Boolean
```

such that

$$(\forall j:0\le j<\Delta:\ S[j] = (a+j \in S))$$

The only operation on W_k^* is of the form x:= next(W_k^*,x). For k>0, this can be done in $\theta(1)$ additions given an array

var WG: **array**[1..nW] **of** 2..g_k

of wheel gaps:

$$WG[1]=2 \ \ \textbf{and}\ \ (\forall i:1<i\le nW:\ WG[i] = W_k^{(i)}-W_k^{(i-1)})$$

The operation p:= next(Primes(p_{k+1}..$\sqrt{N}$),p) can be similarly implemented given an array

var PG: **array**[k+1..np] **of** 2..$d_{\sqrt{N}}$

of prime gaps:

$$(\forall i:k<i\le np:\ PG[i] = p_i-p_{i-1})$$

To avoid multiplications in the code for "sift S", notice that each new value mult = p·f is related to the previous value mult' = p·f' by

$$mult = mult' + p\cdot(next(W_k^*,f')-f')$$

In order to exploit this recurrence, array factor is eliminated in favour of

```
var offset: array[k+1..np] of 0..$g_k\sqrt{N}$
var pWG: array[k+1..np] of 1..nW
```

such that

$$(\forall i{:}k<i\le np{:}\ a+\text{offset}[i] = \min\{p_i\cdot f \mid f\in W_k^* \textbf{ and } p_i\cdot f\ge a\}$$
$$= p_i\cdot f \ \ \textbf{where}\ \ f \textbf{ mod } \Pi_k = W_k^{(pWG[i])})$$

That is, a+offset[i] is the initial value given to mult and pWG[i] is the index of factor[i] in WG. As in the first technique of [18,§6], we use

```
var m: array[1..$g_k/2$] of 0..$g_k\sqrt{N}$
var pm: array[1..$g_k/2$] of 1..$g_k/2$
```

such that each new multiple p·WG[j] replaces the initial zero value in m[WG[j]/2], and the previously computed multiples are simply looked up in m. Array pm is used to contain the indexes WG[j]/2 of the non-zero elements of m, in order to permit fast reinitialization of m.

The space requirement of this implementation is as follows:

S: Δ bits

WG: $\sim nW\cdot \log_2 g_k$ bits

PG: $\sim np\cdot \log_2 d_{\sqrt{N}}$ bits

offset: $\sim np\cdot \log_2(g_k\sqrt{N})$ bits

pWG: $\sim np\cdot \log_2 nW$ bits

m: $\sim \frac{1}{2}g_k \log_2(g_k\sqrt{N})$ bits

pm: $\sim \frac{1}{2}g_k \log_2 g_k$ bits

Strictly, use of the asymptotic indicator "~" is justified only when $\Delta, k \to \infty$ as $N\to\infty$ and $\Pi_k \le N$.

In order to establish the arithmetic complexity of our implementation, we need to appeal to the following number-theoretic facts. (In sums and products, p ranges over primes.)

Fact 1: $\pi(n) \sim n/\log n$ -- the prime number theorem [7, thm. 6]

Fact 2: $p_{i+1} < 2p_i$ -- Bertrand's Postulate [7, thm. 418]

Fact 3: $\sum_{p\leq x} 1/p \sim \log\log x$ -- [7, thm. 427]

Fact 4: $\prod_{p\leq x} (1-1/p) \sim e^{-\gamma}/\log x$ -- Merten's theorem [7, thm. 429]

Fact 5: $\sum_{p\leq x} \log p \sim x$ -- [7, thms. 420,434]

Fact 6: (i) $g_k \geq 2p_{k-1}$ -- [19]

(ii) $g_k = O(p_k^2)$ -- [9]

Since, as is easy to show, $nW = \prod_{i=1}^{k} (p_i-1)$, an immediate corollary of fact 4 is

Fact 4': $nW/\Pi_k \sim e^{-\gamma}/\log p_k$

We now determine the total cost of <u>all</u> executions of the high-level statements in SFWS.

(a) "initialise".

S: $\theta(\Delta/\log N)$ additions, i.e. Δ bit operations.

WG: $\theta(nW)$ additions, *in situ*, using the methods of [18,19].

PG: $\Theta(\sqrt{N}/\log\log\sqrt{N})$ additions, in $\Theta(\sqrt{N}/\log\log\sqrt{N})$ bits, using the sublinear sieve [18]. Alternatively, it can be done in $\Theta(\sqrt{N})$ additions and $O(N^{1/4})$ bits by using the present algorithm recursively. These space/time bounds are used when reducing the overall space requirement in §4.

offset: $\Theta(np)$ additions.

pWG: $\Theta(np)$ additions.

m: $\Theta(g_k)$ additions.

(b) "S:= a..b".

Except for an initial assignment S:= 1..Δ, this can be incorporated in the code for "accumulate the primes in S" by changing the second guarded command in the **if**-statement to

$$y \notin S \rightarrow S := S \cup \{y\}$$

When implemented, the effect is simply to reset each false bit to true.

(c) "sift S".

This abstract statement is performed N/Δ times. Operations referencing the arrays offset and pWG take $\Theta(N/\Delta \cdot np)$ additions. The remaining operations involve deletions. There are $\Theta(1)$ additions per deletion, plus a total number of multiplications to be determined. The number of deletions is that done by Eratosthenes' sieve, reduced by a factor of $\Theta(nW/\Pi_k)$ since factors are taken from W_k^*. So the number of deletions

$$= \Theta(N \cdot \sum_{i=k+1}^{np} 1/p_i) \cdot \Theta(nW/\Pi_k)$$

$$= \Theta(N\log\log N/\log p_k) \quad \text{by facts 3,4'}$$

Since no more than $g_k/2$ different multiples can occur for a given prime on each interval a..b, and at most $O(1)$ multiplications are done per deletion, the number of multiplications

$$\le N/\Delta \cdot \sum_{p \le \sqrt{N}/\log^3 N} g_k/2 \; + \; N \cdot \sum_{\sqrt{N}/\log^3 N < p \le \sqrt{N}} 1/p \cdot \theta(nW/\Pi_k)$$

The first term in this sum[2]

$$= N/\Delta \cdot g_k/2 \cdot \pi(\sqrt{N}/\log^3 N)$$

$$= O\left|\frac{N^{3/2} p_k^2}{\Delta \cdot \log^4 N}\right| \quad \text{by facts 1,6(ii)}$$

The second term

$$\sim N \cdot (\log\log\sqrt{N} - \log\log(\sqrt{N}/\log^3 N)) \cdot e^{-\gamma}/\log p_k \quad \text{by facts 3,4'}$$

$$\sim N \cdot \log(1 - 3\log\log N/\log\sqrt{N})^{-1} \cdot e^{-\gamma}/\log p_k$$

$$= \theta\left|\frac{N \cdot \log\log N}{\log p_k \cdot \log N}\right| \quad \text{since } \log(1-\epsilon)^{-1} \to \epsilon \text{ as } \epsilon \to 0$$

(d) "accumulate the primes in a..b".

This abstract statement is also done N/Δ times. Since each member of W_k^* is examined in $\theta(1)$ additions, the total cost

$$= N \cdot \theta(nW/\Pi_k) \text{ additions}$$

$$= \theta(N/\log p_k) \text{ additions} \quad \text{by fact 4'}$$

We can now establish the claim that choosing $(\Delta, k) = (\sqrt{N}, \max\{k \mid \Pi_k \le \sqrt{N}\})$ gives a compact linear sieve. We have

$$np \sim \sqrt{N}/\log\sqrt{N} \quad \text{by fact 1}$$

$$p_k = \theta(\log N) \quad \text{by facts 2,5}$$

[2]If the second technique in [18,§6] is used for $p \le \sqrt{N}/\log^3 N$, it is only necessary to do this many additions.

$nW = O(\sqrt{N}/\log\log N)$ by fact 4'

$\log_2 g_k = \Theta(\log\log N)$ by fact 6

Since $d_{\sqrt{N}} = O(\sqrt{N})$, the storage requirement of SFWS(Δ,k) is seen to be $\Theta(\sqrt{N})$ bits, with the storage for array offset being critical. The arithmetic complexity breaks down as follows. Note that the operations required for <u>all</u> executions of the abstract statements are given.

"initialise": $\Theta(\sqrt{N}/\log\log N)$ additions.

"S:= a..b" and "accumulate the primes in S": $\Theta(N/\log\log N)$ additions.

"sift S": manipulating offset and pWG: $\Theta(N/\log N)$ additions; performing deletions: $\Theta(N)$ additions and $\Theta(N/\log N)$ multiplications.

Since a multiplication can be simulated in $\Theta(\log N)$ additive operations by the well-known "shift and add" method, the arithmetic complexity amounts to $\Theta(N)$ additions.

A similar calculation shows that the segmented sieve of Eratosthenes [2,17], i.e. SFWS with $(\Delta,k) = (\sqrt{N},0)$, requires $\Theta(\sqrt{N})$ bits and $\Theta(N\cdot\log\log N)$ additions.

4. Further Reducing the Storage Requirement

The extra factor of $\Theta(\log\log N)$ in the complexity of the segmented sieve of Eratosthenes actually allows Δ to be reduced to $\Theta(\sqrt{N}/\log\log\sqrt{N})$, without increasing the complexity. Also, the array offset can be dispensed with by recalculating instead of saving. This involves $\Theta(N/\Delta\cdot np) = \Theta(N\cdot\log\log N/\log N)$ multiplications, easily simulated with $\Theta(N\cdot\log\log N)$ additions. So the storage requirement reduces to $\Theta(\sqrt{N}/\log\log N)$ bits if the prime gaps in PG require no more than that. This is certainly the case if $d_n = \Theta(\log^2 n)$, as is strongly conjectured. However, the best result to date is $d_n = O(n^{0.55+\epsilon})$ for any $\epsilon > 0$ [8], which is not nearly strong enough.

Consider instead the following scheme. Rather than storing a prime gap $p_i - p_{i-1}$ in an integer array element PG[i], store the binary representation of $p_i - p_{i-1}$, <u>without leading zeros</u>, in a bit array; also use an integer array nbits such that nbits[i] = the number of bits stored for that gap. Since the primes are taken in sequence, the next prime can still be calculated in $\Theta(1)$ additive operations. Since, by fact 1, the <u>average</u> prime gap is logN, the bit array requires $O(np \cdot \log\log N) = O(\sqrt{N} \cdot \log\log N/\log N)$ bits, and array nbits requires $O(np \cdot \log\log d_{\sqrt{N}}) = O(\sqrt{N} \cdot \log\log N/\log N)$ bits. So this method provably does the job[3].

By resorting to the (asymptotically) fast multiplication scheme of Schönhage and Strassen (see [1]), Δ can be reduced to $\Theta(\sqrt{N} \cdot \log\log\log N/\log N)$ and the segmented sieve SFWS(Δ,0) still implemented in $\Theta(N \cdot \log\log N)$ additions without using array offset. The reason is that the $\Theta(N/\Delta \cdot np)$ multiplications required amount to $\Theta(N \cdot \log\log N)$ additions when each multiplication is counted as $\Theta(\log\log N \cdot \log\log\log N)$ additions. However, the compacted prime gaps now dominate the storage requirement, which is $O(\sqrt{N} \cdot \log\log N/\log N)$ bits.

Dispensing with the arrays offset and pWG in the compact linear sieve involves $\Theta(N/\Delta \cdot np)$ calculations of the minimum f such that

[3]If $d_n = \Theta(\log^2 n)$ then array nbits needs only $\Theta(\sqrt{N} \cdot \log\log\log N/\log N)$ bits. We do not know if the bit array uses only $o(\sqrt{N} \cdot \log\log N/\log N)$ bits.

$$f \ \mathbf{mod}\ \Pi_k \in W_k \quad \mathbf{and} \quad p_i \cdot f \geq a,$$

whence the required offset is just $p_i \cdot f - a$, and the corresponding index in WG is that i such that

$$f \ \mathbf{mod}\ \Pi_k = W_k^{(i)}$$

Each calculation can be done in $\Theta(1)$ multiplications by first setting $v = \lceil a/p_i \rceil$ and $w = v \ \mathbf{mod}\ \Pi_k$. Then $f = v + (\mathrm{next}(W_k^*, w-1) - w)$ and the index in WG is that of $\mathrm{next}(W_k^*, w-1)$. To handle these new kinds of operations on W_k in $\Theta(1)$ additions, we can use

```
var WGto: array[0..$\Pi_k$-1] of 2..$g_k$
var Windex: array[1..$\Pi_k$] of 0..nW
```

such that

$$(\forall i{:}0 \leq i < \Pi_k\text{: } \mathrm{WGto}[i] = \mathrm{next}(W_k^*, i) - i) \quad \mathbf{and}$$
$$(\forall j{:}1 \leq j \leq nW\text{: } \mathrm{Windex}[W_k^{(j)}] = j)$$

Note that array WGto obviates the need for array WG, and that both WGto and Windex can be created in $\Theta(\Pi_k)$ additions by adapting the sublinear sieve.

The information in these new arrays also permits a compaction in S, because the SFWS algorithm only refers to $S \cap W_k^*$. So the storage for S can be reduced to $\sim \Delta \cdot e^{-\gamma}/\log p_k$ without increasing the order of complexity. Let SFWSC(Δ,k) denote such a Segmented Fixed-Wheel Sieve with Compaction. Since $\Pi_k = \Theta(N^c)$ for any $c > 0$ suffices to give a linear algorithm, arrays WGto and Windex need take no more space than S. Thus SFWSC with $(\Delta,k) = (\sqrt{N}, \max\{k \mid \Pi_k \leq \sqrt{N}/\log^2 N\})$, for instance, can be implemented in $\Theta(\sqrt{N}/\log\log N)$ bits to take $\Theta(N)$ additions. Similarly, using fast multiplication, SFWSC with $(\Delta,k) = (\sqrt{N} \cdot \log\log N \cdot \log\log\log N/\log N, \max\{k \mid \Pi_k \leq \sqrt{N}/\log^2 N\})$, can be implemented to take $\Theta(N)$ additions and run in $O(\sqrt{N}\cdot\log\log N/\log N)$ bits (the storage required for the prime gaps).

5. Some Actual Figures

With $N = 10^6$, an optimized version of SFWS with $(\Delta,k) = (10^3,6)$ performed 177765 deletions, saved values in array offset 113734 times, and used 126907 multiplications. Using Pascal's packed arrays [10], the storage required for all arrays on a 36 bit machine was 181 words. With $(\Delta,k) = (10^3,1)$ -- the usual version of the segmented sieve of Eratosthenes [2] -- 849047 deletions were performed. This clearly shows the efficacy of our exploitation of a fixed wheel. With $N = 10^{12}$ and $(\Delta,k) = (10^6,7)$ the storage requirement would be 149829 words, a practical proposition.

6. Conclusions

The ideas of sifting the integers in _segments_, and of using a _wheel_, have been shown to underly many of the published algorithms for finding the primes up to N (albeit implicitly in the case of the latter idea). Even Dijkstra's algorithm can be seen to stem from a (pathological) version of a segmented sieve, making sense of his claim to have "made our computation close to an implementation of the Sieve of Eratosthenes!" [4, p.38]. (This despite the huge difference in complexity; see [17].)

The construction of linear, additive, prime number sieves running in $O(\sqrt{N})$ bits shows very clearly the power of wheels -- compare the very complicated storage reduction for a linear _multiplicative_ sieve in [15], which still does not obtain a space requirement of $O(N)$ bits!

Without using fast multiplication, prime number sieves are now known with the following time and space requirements:

- $\Theta(N/\log\log N)$ additions in $\Theta(N/\log\log N)$ bits -- [18]
- $\Theta(N)$ additions in $\Theta(\sqrt{N}/\log\log N)$ bits
  -- $SFWSC(\sqrt{N}, \max\{k \mid \Pi_k \le \sqrt{N}/\log^2 N\})$

Permitting fast multiplication, we have:

- $\Theta(N)$ additions in $O(\sqrt{N}\cdot\log\log N/\log N)$ bits
  -- $SFWSC(\sqrt{N}\cdot\log\log N\cdot\log\log\log N/\log N, \max\{k \mid \Pi_k \le \sqrt{N}/\log^2 N\})$

Notice the large difference in the space requirements of the sublinear and linear sieves; it would be interesting to know whether or not the former can be reduced. Reducing the latter would seem to require fresh insights, as our implementations are very finely tuned.

We close with two observations. The first is that parallelization of the SFWS sieves is straightforward -- the idea is simply to assign each of N/Δ processors to a segment of length Δ. See [16] for a methodical exposition. The second is that the fact tnat no fixed wheel sieve is as efficient as the sublinear sieve further highlights the elegance of the latter.

Acknowledgements

This paper may not have been written but for the stimulating enthusiasm of Ian Parberry. I also profited from discussions with Richard Brent. The paper profited from the scrupulous scrutiny of David Gries.

References

[1] Aho, A., J. Hopcroft and J. Ullman. The Design and Analysis of Computer Algorithms. Addison-Wesley, Reading, Massachusetts, 1974.

[2] Bays, C. and R. H. Hudson. The segmented sieve of Eratosthenes and primes in arithmetic progression to 10^{12}. B.I.T. 17 (1977), 121-127.

[3] Brent, R.P. The first occurrence of large gaps between successive primes. Math. Comp. 27, 124 (October 1973), 959-963.

[4] Dijkstra, E.W. Notes on structured programming. In Dahl, O.J., C.A.R. Hoare and E.W. Dijkstra: Structured Programming. Academic Press, New York, 1972, 1-82.

[5] Dijkstra, E.W. Guarded commands, nondeterminacy and formal derivation of programs. Comm. ACM 18, 8 (August 1975), 453-457.

[6] Gries, D. and J. Misra. A linear sieve algorithm for finding prime numbers. Comm. ACM 21, 12 (December 1978), 999-1003.

[7] Hardy, G.H. and E.M. Wright. An Introduction to the Theory of Numbers. 5th Ed., Oxford University Press, Oxford, England, 1979.

[8] Heath-Brown, D.R. and H. Iwaniec. On the difference between consecutive primes. Inventiones Mathematicae 55, 1 (December 1979), 49-69.

[9] Iwaniec, H. On the problem of Jacobsthal. Demonstratio Math. 11, (1978), 225-231.

[10] Jensen, K. and N. Wirth. Pascal - User Manual and Report. Springer-Verlag, Berlin and New York, 1974.

[11] Knuth, D.E. The Art of Computer Programming, vol. 2: Seminumerical Algorithms. 2nd Ed., Addison-Wesley, Reading, Massachusetts, 1981.

[12] Knuth, D.E. The Art of Computer Programming, vol. 3: Sorting and Searching. Addison-Wesley, Reading, Massachusetts, 1973.

[13] LeVeque, W.J. Elementary Theory of Numbers. Addison-Wesley, Reading, Massachusetts, 1965.

[14] Mairson, H.G. Some new upper bounds on the generation of prime numbers. Comm. ACM 20, 9 (September 1977), 664-669.

[15] Misra, J. Space-time tradeoff in implementing certain set operations. Inf. Proc. Letters 8, 2 (February 1979), 81-85.

[16] Parberry, I. Unpublished manuscript on fast parallel prime-number sieves. Dept. of Computer Science, University of Queensland, 1981.

[17] Pritchard, P. On the prime example of programming. In Language Design and Programming Methodology: Lecture Notes in Computer Science 79, Springer-Verlag, Berlin Heidelberg and New York, 1980, 85-94.

[18] Pritchard, P. A sublinear additive sieve for finding prime numbers. Comm. ACM 24, 1 (January 1981), 18-23.

[19] Pritchard, P. Explaining the wheel sieve. Submitted.