Journal of Discrete Mathematical Sciences & Cryptography
ISSN 0972-0529 (Print), ISSN 2169-0065 (Online)
Vol. 26 (2023), No. 4, pp. 1213–1236
DOI : 10.47974/JDMSC-1629

A new algorithm to find prime numbers with less memory requirements

Daniele Bufalo
Department of Informatics
Università degli Studi di Bari Aldo Moro
Via Orabona 4
Bari, I-70125
Italy

Michele Bufalo[†]
Department of Methods and Models for Economics
Università degli Studi di Roma "La Sapienza"
Territory and Finance
Via del Castro Laurenziano 9
Roma, I-00185
Italy

Giuseppe Orlando[*]
Department of Economics and Finance
Università degli Studi di Bari Aldo Moro
Largo Abbazia S. Scolastica
Bari, I-70124
Italy

and

Departamento de Matemáticas
Universitad de Jaén
Campus Las Lagunillas, s/n
Jaén S-23071
Spain

and

[†] *E-mail:* michele.bufalo@uniroma1.it
[*] *E-mail:* giuseppe.orlando@uniba.it (Corresponding Author)

HSE University
Department of Economics
Soyuza Pechatnikov Street 16
Saint Petersburg, R-190121
Russia

Raffaele Tetta
Department of Informatics
Università degli Studi di Bari Aldo Moro
Via Orabona 4
Bari, I-70125
Italy

Abstract

In this paper, we suggest a new approach to find any prime numbers up to a given $n \in \mathbb{N}^*$. The proposed procedure does not work like a sieve and is easy to implement as it only uses assignments and subtractions which lead to improvements in memory requirements and upgradeable runtime performance. This is because, also, the algorithm suits well parallel computing. These results aim to solve several problems affecting those routines based on sieve methods, especially when large numbers are considered.

Subject Classification: *[2010] 11A41, 11Y16.*

Keywords: *Hill cipher, Public key, Prime numbers, Factorization, Prime numbers, Sieve method.*

1. Introduction

To date the distribution of prime numbers is unknown and the most advanced attempt is the Riemann conjecture [4]). For this reason, prime numbers play a crucial role in cryptography. For example, some important cryptographic algorithms, such as the RSA algorithm (see [10]), depending on the prime factorization of large integers. A *public key* consists of a product of two large primes used to encrypt a message, while a *secret key* consists of those two primes used to decrypt the message. Therefore, the security of these algorithms is based on the key factorization, which is non-trivial if large and unknown primes are involved. As secret keys are regenerated after few seconds, there is no sufficient time for discovering the keys by factoring large integers. Thus, the security of these algorithms is related to the problem of the modular inverse, which also depends on the prime factorization (see, e.g., [6], for recent developments in this field).

As increasingly bigger numbers are used to increase information security they impose strain on memory requirements. The largest known prime, called $M82589933$, is a 24-million-digit number that does not allow encryption on common computers, due to memory limitation.

In absence of knowledge about the distribution of primes, their search has been conducted with the sieve of Eratosthenes and its extensions, such as the sieve of Atkin or the sieve of Sundaram (see, e.g., [3] and [1]). On a different path, Trigiante et al. [14] defined a sieve that permits to estimate the number of primes less than an arbitrary N by solving a difference equation. These methods are efficient due to their *pseudo– polynomial* time complexity, but they generally require $O(n)$ of memory, which is the reason why they get slow when n is large. Many solutions for this problem have been proposed in the literature, such as the *segmented* sieves which, even working with the same operations as the classic sieve, reduce the memory requirement. However, when n is very large the sieving complexity becomes $O(\sqrt{n})$ and may not fit in memory. This limitation drives improvements in supercomputing for decrypting and data mining. Indeed, modern hardware systems consist of dozens of high-end consumer graphics cards, employed for solving thousands of calculations. The problem is compounded by the fact that, as explained in Section 2, the sieves have a lot of issues if implemented by parallel computing. In fact, just a few primes with millions of digits have been listed with the parallel procedure. To date, to find a new one big prime, modern algorithms apply some primality test (see e.g. the Lucas-Lehmer test [9], or the AKS test [2]) to some big Mersenne numbers, i.e., numbers of type $2^p - 1$, where p is a given (large) prime (for more details on Mersenne numbers, see [12]). Similar tests are slow for numbers that have more than several million digits due to the complexity of each operation involved, which grows with the number of digits. Among the approaches aimed at exploring sets of prime numbers that identify patterns in their structure, we can mention supervised learning models such as Support Vector Domain Description [5].

Alternative cryptographic methods are based on polygraphic substitution where cipher is based on a uniform substitution on blocks of letters. Early cipher in which the substitution involved pairs of letters (i.e. bigraphic substitution) was devised by Giambattista della Porta in 1563 [8]. Modern cryptography requiring matrix algebra to encrypt blocks of any desired length and computers was introduced by Lester S. Hill in 1929 [7]. However, being the Hill cypher linear, decipher takes little time. Improvements have been suggested over the years. For examples,

Viswanath et al. [15] suggested a robust public key cryptosystem and Sundarayya [13] increased the security using Affine Hill Cipher involving two or more digital signatures under modulation of prime number. Thus, once again, the key issue in cryptography is related to prime numbers. For this reason, the purpose of the present article is to suggest a new algorithm for finding large prime numbers using less memory and which could be easily implemented in parallel processing. Furthermore, we show that the algorithm provides prime factorization for any given integer.

The paper is divided as follows: in Section 2 we recall the classic sieve of Eratosthenes, which is taken as a benchmark. Section 3 describes the proposed algorithm and proves its correctness. Sections 4 and 5 present optimizations and generalizations. Section 6 reports the obtained numerical tests and adds further considerations to the computations. Then, it compares the suggested algorithm with the sieve of Eratosthenes, its segmented version and the sieve of Atkin. Finally, Section 7 concludes.

2. The sieve of Eratosthenes and its extensions

One of the most famous algorithms for generating prime numbers is the ancient sieve of Eratosthenes. Fixed $n \in \mathbb{N}^*$, it consists of the following steps:

1. Create a list of consecutive integers from 2 through n;
2. Mark any multiple of each (unmarked) integer $i \in \{2, 3, ..., \lfloor \sqrt{n} \rfloor\}$ of the list (where $\lfloor \sqrt{n} \rfloor$ denotes the (floor) ceiling part of $\sqrt{n}$), starting from i^2.

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100

Figure 1

Prime numbers up to 100, computed by the sieve of Eratosthenes

At the end of loop 2, all the unmarked numbers are prime (see Figure 1). We note that the sieve of Eratosthenes is not practical for identifying big primes since its complexity is $\mathcal{O}(n \ln(n))$ with a memory requirement

of $\mathcal{O}(n)$ (where n is exponential in the number of digits). The main problem with the sieve of Eratosthenes is not the number of operations that it performs but rather its memory requirements. For large n, the range of primes may not fit in memory. In the literature, there are many optimizations of this method, such as the segmented sieve of Eratosthenes, which working with the same operations of the classic sieve has $\mathcal{O}(\sqrt{n})$ space complexity, or the sieve of Atkin, which uses $\mathcal{O}(n)$ operations with only $n^{(0.5+\mathcal{O}(1))}$ bits of memory (for more details, see [3]).

In particular, the sieve of Eratosthenes becomes to be hardly implemented in parallel processing. Indeed, the parallel implementation suffers the following issues[1]:

1. Two processors may asynchronously end up using the same prime to sieve
   - The first processor will access the value of the current prime and start sieving with it,
   - The second processor will start from the next unmarked cell, which it updates as the current prime,
   - If another processor starts before this update then it also starts sieving with the same prime;
2. Also a processor may end up sieving with composite numbers
   - The first processor starts sieving with multiples of 2,
   - Before it marks any cell, a second processor starts sieving with the next prime, which is 3,
   - The third processor starts sieving with the next unmarked cell, which is 4 (and has not been marked yet by the first processor).

For the reasons said above, the parallel implementation needs some manipulations (see, e.g. [16]).

3. The new proposed algorithm

3.1 *Preliminary results*

In order to introduce the proposed new algorithm, we illustrate some properties about prime numbers. Through this Section, we let

[1] See PRAM Model of Parallel Computation: http://cse.iitkgp.ac.in/ debdeep/courses_iit-kgp/PAlgo/index.htm;

$\mathbb{D} = \{i \in \mathbb{N}^* \mid i = 2n+1, n \in \mathbb{N}^*\}$ be the set of the odd integers greater than or equal to 3, and $\mathcal{P}$ the subset of $\mathbb{D}$ consisting only of primes.

Definition 3.1 : For any $i \in \mathbb{D}$, define the quantity $C^{(i)}$ as follows

$$C^{(i)}(j) = \left[\frac{\left(\left\lfloor \frac{i}{P(j)} \right\rfloor + \xi^{(i)}(j) \right) P(j) - i}{2} \right]_{P(j)}, \qquad 1 \leq j \leq m, \tag{1}$$

where

$$\xi^{(i)}(j) = \begin{cases} 2 & \text{if } 2 \left| \left\lfloor \frac{i}{P(j)} \right\rfloor \right. \\ 1 & \text{otherwise,} \end{cases}$$

$$m = \max\{j \in \mathbb{N}^* \mid P(j) \leq i, P(j) \in \mathcal{P}\} \tag{2}$$

and $[\cdot]_p$ denotes the residue modulo P.

In other words, $C^{(i)}$ is a list whose j^{th}-component represents the half-distance (modulo $P(j)$) between the first (odd) multiple of $P(j)$, greater than i, and i itself.

Example 3.2 : Set $i = 39$, the primes in $\mathcal{P}$ below i are $\{3,5,7,11,13,17,19,23,29,31,37\}$ and the list $C^{(39)}$ is given by

$$C^{(39)} = [0,3,5,8,0,6,9,15,24,27,36].$$

Indeed, for instance, if $j = 4$ $(P(4) = 11)$ we have that 55 is the first (odd) multiple of 11, with $55 > 39$, and the half-distance between 55 and 39 is just 8.

Notice that $C^{(i)}(j)$ is null when $P(j)$ divides i, for any j. This fact is looked into the next proposition.

Proposition 3.3 : Consider an integer $i \in \mathbb{D}$, and let $C^{(i)}$ be the quantity defined in (1). Then

$$i \in \mathcal{P} \Leftrightarrow C^{(i)}(j) \neq 0, \forall j \leq m, \tag{3}$$

where m is given by (2).

Proof: Fixed j, we have that $C^{(i)}(j) \neq 0$ it is equivalent to say there exists no $q \in \mathbb{N}^*$ such that

$$i = \left(\left\lfloor \frac{i}{P(j)} \right\rfloor + \xi^{(i)}(j) - 2q \right) P(j),$$

i.e., i it is not a multiple of $P(j)$. Hence, if this holds true for any $j \leq m$, it is equivalent to state that i it is not a multiple of any primes (less than i), or equivalently, $i \in \mathcal{P}$. □

Corollary 3.4 : *Fixed* $j \leq m$, *one has*

$$C^{(i)}(j) = 0 \Leftrightarrow P(j) \mid i.$$

Now, according to the Example 3.2, we may soon deduce that 39 is a not prime, since $C^{(39)}(1) = 0$. This obvious consideration leads us to simplify Definition 3.1.

Definition 3.5 : For any integer $i > 3$, define the quantity $C^{(i)}$ as

$$C^{(i)}(j) = \left[\frac{\left(\left\lfloor \frac{i}{P(j)} \right\rfloor + \xi^{(i)}(j) \right) P(j) - i}{2} \right]_{P(j)} \qquad 1 \leq j \leq m, \tag{4}$$

where

$$\xi^{(i)}(j) = \begin{cases} 2 & \text{if } \left\lfloor \frac{i}{P(j)} \right\rfloor \\ 1 & \text{otherwise,} \end{cases}$$

and

$$l = \max\{ j \in \mathbb{N}^* \mid P(j) < \lfloor \sqrt{i} \rfloor + 1, P(j) \in \mathcal{P} \}. \tag{5}$$

The above modification allows us to enhance Proposition 3.3.

Proposition 3.6 : Consider an integer $i \in D - \{3\}$, and let $C^{(i)}$ be the list defined in (4). Then

$$i \in \mathcal{P} \Leftrightarrow C^{(i)}(j) \neq 0, \forall j \leq l, \tag{6}$$

where l is given by (5).

Proof: The proof is based on that of Proposition 3.3. It remains to show the sentence "$\forall j \leq l$" in (6). Indeed, if there exists $h > l$ such that $P(h)$ divides

i, one has $i = qP(h)$ for a certain $q \in \mathbb{N}^*$. Thus, we have to distinguish two cases.

(i) If $q \in \mathcal{P}$, then it must satisfies the inequalities $q < \sqrt{i}$, as observed in the sieve of Eratosthenes. This fact forces q to be equal to $P(j)$, with $j < l$, that means $P(j) | i$. But this is absurd under the assumption $C^{(i)}(j) \neq 0$ for $j < l$.

(ii) Otherwise, assume there exists q' such that $q = q'P(j)$. Again, we observe that $P(j) < \sqrt{q}$, which implies $P(j) < \sqrt{i}$, that is $j < l$. Arguing as in i), this statement is absurd.

This concludes the proof. □

In view of Proposition 3.6, one might wonder the reason why we set

$$P(j) < \lfloor \sqrt{i} \rfloor + 1 \tag{7}$$

in (5), instead of

$$P(j) < \lfloor \sqrt{i} \rfloor. \tag{8}$$

The next counterexample clarifies this question.

Example 3.7 : Take i as a perfect square, e.g., $i = 25$. By Definition 3.1, we have

$$C^{(25)} = [1, 0, 5, 4, 7, 13, 16, 22].$$

Since the second component is null we may conclude that $P(2)(= 5)$ divides 25, i.e., $25 \notin \mathcal{P}$. Analogously, according to Definition 3.5, we can write

$$C^{(25)} = [1, 0];$$

and it is still evident that $25 \notin \mathcal{P}$. On the contrary, the assumption (8) implies $C^{(25)} = [1]$, and one could conclude that $25 \in \mathcal{P}$!

Therefore, the Example 3.7 suggests us to pay attention to the perfect squares, which must be treated as a special case.

3.2 *The algorithm*

The aim of the proposed algorithm is to search for every prime number below a fixed integer n greater than 3. Recursively, at each iteration

$i \in \mathbb{D}, i \leq n$, we assess if the number i is prime or not by computing the list $C^{(i)}$ presented in Definition 3.5. By Proposition 3.6, we establish if $i \in \mathcal{P}$, and, if it is true, we update a second list, named P, containing the primes found until the iteration i.

At the first iteration, we set $i = 3$ and initialize $C^{(3)} = [3]$ and $P(1) = 3$. From Definition 3.5, it is clear that $C^{(5)} = [2]$ and $P(2) = 5$, $C^{(7)} = [1]$ and $P(3) = 7$, and so on.

More generally, given $C^{(i)}$, in order to compute the next $C^{(i+2)}$ we use the helpful Proposition 3.9, which is more convenient than Definition 3.5; after recalling the following lemma.

Lemma 3.8 : *Given* $a, b \in \mathbb{N}^*$, *we have*

$$\left\lfloor \frac{a}{b} \right\rfloor - \left\lfloor \frac{a-1}{b} \right\rfloor = \begin{cases} 1 & \text{if } b \mid a \\ 0 & \text{otherwise.} \end{cases}$$

Proposition 3.9 : For any $i \in \mathbb{D}$ one has

$$C^{(i+2)}(j) = [C^{(i)}(j) - 1]_{P(j)}, \tag{9}$$

for any $j \leq l$.

Proof: We must distinguish three cases:

$$i)\ P(j) \mid (i+2), \qquad ii) \begin{cases} P(j) \nmid (i+2) \\ P(j) \mid (i+1), \end{cases} \qquad iii) \begin{cases} P(j) \nmid (i+2) \\ P(j) \nmid (i+1), \end{cases} \tag{10}$$

for any prime $P(j) \leq \lfloor \sqrt{i+2} \rfloor + 1$.

(i) By virtue of Lemma 3.8, if $P(j) \mid (i+2)$ then

$$\left\lfloor \frac{i+2}{P(j)} \right\rfloor = \left\lfloor \frac{i}{P(j)} \right\rfloor + 1;$$

moreover, since $2 \left| \left(\frac{i+2}{P(j)} \right) \right.$, it is easy to see that $\xi^{(i+2)}(j) = 2$ and $\xi^{(i)}(j) = 1$. Hence,

$$C^{(i+2)}(j) = \left[\frac{\left(\left\lfloor \frac{i+2}{P(j)} \right\rfloor + \xi^{(i+2)}(j) \right) P(j) - i - 2}{2} \right]_{P(j)} = \left[\frac{\left(\left\lfloor \frac{i}{P(j)} \right\rfloor + 1 + 2 \right) P(j) - i - 2}{2} \right]_{P(j)} =$$

$$\left[\frac{\left(\left\lfloor \frac{i}{P(j)} \right\rfloor + \xi^{(i)}(j) \right) P(j) - i}{2} + P(j) - 1 \right]_{P(j)} = [C^{(i)}(j) - 1]_{P(j)}.$$

(ii) If $p \nmid (i+2)$ and $p \mid (i+1)$, we have that

$$\left\lfloor \frac{i+2}{P(j)} \right\rfloor = \left\lfloor \frac{i}{P(j)} \right\rfloor + 1,$$

by Lemma 3.8, and

$$\xi^{(i+2)}(j) = 1, \qquad \xi^{(i+1)}(j) = 1, \qquad \xi^{(i)}(j) = 2,$$

since $2 \left| \left(\frac{i+1}{P(j)} \right) \right.$. Therefore,

$$C^{(i+2)}(j) = \left[\frac{\left(\left\lfloor \frac{i+2}{P(j)} \right\rfloor + \xi^{(i+2)}(j) \right) P(j) - i - 2}{2} \right]_{P(j)} = \left[\frac{\left(\left\lfloor \frac{i}{P(j)} \right\rfloor + 1 + 1 \right) P(j) - i - 2}{2} \right]_{P(j)} =$$

$$\left[\frac{\left(\left\lfloor \frac{i}{P(j)} \right\rfloor + \xi^{(i)}(j) \right) P(j) - i}{2} - 1 \right]_{P(j)} = [C^{(i)}(j) - 1]_{P(j)}.$$

(iii) If $p \nmid (i+2)$ and $p \nmid (i+1)$, we have

$$\left\lfloor \frac{i+2}{P(j)} \right\rfloor = \left\lfloor \frac{i}{P(j)} \right\rfloor,$$

by Lemma 3.8, and $\xi^{(i+2)}(j) = \xi^{(i)}(j)$. So that,

$$C^{(i+2)}(j) = \left[\frac{\left(\left\lfloor \frac{i+2}{P(j)} \right\rfloor + \xi^{(i+2)}(j) \right) P(j) - i - 2}{2} \right]_{P(j)} =$$

$$\left[\frac{\left(\left\lfloor \frac{i}{P(j)} \right\rfloor + \xi^{(i)}(j) \right) P(j) - i}{2} - 1 \right]_{P(j)} = [C^{(i)}(j) - 1]_{P(j)}.$$

□

Remark 3.10 : With reference to formula (9), the authors suggest avoiding the congruence modulo $P(j)$. Indeed, one may set, recursively,

$$C^{(i+2)}(j) = \begin{cases} P(j) & \text{if } C^{(i+2)}(j) = 0, \\ C^{(i)}(j) - 1 & \text{otherwise,} \end{cases}$$

in order to reduce the computational cost. In this way, the algorithm works just with subtractions and assignments.

Furthermore, as highlighted in Example 3.7, we have to increase the length of $C^{(i)}$ when $i = P^2(l)$. The perfect square of a prime is identified from the following criterion.

Proposition 3.11 : For any $i \in \mathbb{D} - \{3\}$, let l be the size of $C^{(i-2)}$. If

$$\begin{cases} C^{(i)}(j) \neq 0 & \text{if } j \neq l \\ C^{(i)}(l) = 0, \end{cases}$$

then $i = P^2(l)$.

Proof: Due to the Corollary 3.4, if $C^{(i)}(j) \neq 0$ for any $j \neq l$, and $C^{(i)}(l) = 0$, then i may be multiple only of $P(l)$, so that $i = P^2(l)$. Indeed, $i \neq P(l)$ and i can not be multiple of another prime $P(h)$, $h > l$, from Definition 3.5. □

Again, we might compute $C^{(i)}(l+1)$ by the (3.5), or, to keep this simple, by the next formula.

Proposition 3.12 : If $i = P^2(l)$, for fixed $l \leq m$, we have

$$C^{(i)}(l+1) = \left[P(l+1) - \frac{d^2}{2} \right]_{P(l+1)}, \tag{11}$$

where $d = P(l+1) - P(l)$.

Proof: Applying relation (9) repeatedly, we soon obtain

$$C^{(i)}(l+1) = \left[C^{(P(l+1))}(l+1) - \frac{P^2(l) - P(l+1)}{2} \right]_{P(l+1)} = \left[P(l+1) - \frac{d^2}{2} \right]_{P(l+1)};$$

where $P^2(l) - P(l+1)$ represents the distance between i and $P(l+1)$, and it is equivalent to d^2 (modulo $P(l+1)$). □

Note that the square d^2 is fast to compute also for large l, since there exist infinitely pairs of (consecutive) primes whose gap is less than $7 \cdot 10^7$ (see [17]).

Summing up, the scheme of this algorithm is described below.

1. Initialize $i = 3$, $j = 1$, $P(j) = 3$ and $C^{(i)}(j) = 3$.
2. While $i < n$, increment $i = i + 2$.
   - While $j \leq l$ (where l is the size of $C^{(i-2)}$), let $C^{(i)}(j)$ equal to

     $$C^{(i)}(j) = \begin{cases} C^{(i-2)}(j) - 1 & \text{if } C^{(i-2)}(j) \neq 1, \\ P(i) & \text{otherwise,} \end{cases} \tag{12}$$

     according to Remark 3.10.

- If there exists $j \le l$ such that $C^{(i)}(j) = 0$, then $i \notin \mathcal{P}$. In particular, if only $C^{(i)}(l) = 0$, then the current index i is the square of $P(l)$, and we set

$$C^{(i)}(l+1) = P(l+1) - \frac{(P(l+1) - P(l))^2}{2}.$$

- Else, if $C^{(i)}(j) \neq 0$, for any $j \le l$, then the index $i \in \mathcal{P}$ and update P.

$i=3$	$i=5$	$i=7$	$i=9$	$i=11$	$i=13$	$i=15$	$i=17$
$C^{(3)}$ [3]	$C^{(5)}$ [2]	$C^{(7)}$ [1]	$C^{(9)}$ [$\underset{\to 3}{0}$, 3]	$C^{(11)}$ [2, 2]	$C^{(13)}$ [1, 1]	$C^{(15)}$ [$\underset{\to 3}{0}$, $\underset{\to 5}{0}$]	$C^{(17)}$ [2, 4]
$P(1) = 3$	$P(2) = 5$	$P(3) = 7$	–	$P(4) = 11$	$P(5) = 13$	–	$P(6) = 17$

$i=19$	$i=21$	$i=23$	$i=25$	$i=27$	$i=29$	$i=31$
$C^{(19)}$ [1, 3]	$C^{(21)}$ [$\underset{\to 3}{0}$, 2]	$C^{(23)}$ [2, 1]	$C^{(25)}$ [1, $\underset{\to 5}{0}$, 5]	$C^{(27)}$ [$\underset{\to 3}{0}$, 4, 4]	$C^{(29)}$ [2, 3, 3]	$C^{(31)}$ [1, 2, 2]
$P(7) = 19$	–	$P(8) = 23$	–	–	$P(9) = 29$	$P(10) = 31$

$i=33$	$i=35$	$i=37$	$i=39$	$i=41$	$i=43$
$C^{(33)}$ [$\underset{\to 3}{0}$, 1, 1]	$C^{(35)}$ [2, $\underset{\to 5}{0}$, $\underset{\to 7}{0}$]	$C^{(37)}$ [1, 4, 6]	$C^{(39)}$ [$\underset{\to 3}{0}$, 3, 5]	$C^{(41)}$ [2, 2, 4]	$C^{(43)}$ [1, 1, 3]
–	–	$P(11) = 37$	–	$P(12) = 41$	$P(13) = 43$

$i=45$	$i=47$	$i=49$	$i=51$	$i=53$	$i=55$
$C^{(45)}$ [$\underset{\to 3}{0}$, $\underset{\to 5}{0}$, 2]	$C^{(47)}$ [2, 4, 1]	$C^{(49)}$ [1, 3, $\underset{\to 7}{0}$, 3]	$C^{(51)}$ [$\underset{\to 3}{0}$, 2, 6, 2]	$C^{(53)}$ [2, 1, 5, 1]	$C^{(55)}$ [1, $\underset{\to 5}{0}$, 4, $\underset{\to 11}{0}$]
–	$P(14) = 47$	–	–	$P(15) = 53$	–

$i=57$	$i=59$	$i=61$	$i=63$	$i=65$	$i=67$
$C^{(57)}$ [$\underset{\to 3}{0}$, 4, 3, 10]	$C^{(59)}$ [2, 3, 2, 9]	$C^{(61)}$ [1, 2, 1, 8]	$C^{(63)}$ [$\underset{\to 3}{0}$, 1, $\underset{\to 7}{0}$, 7]	$C^{(65)}$ [2, $\underset{\to 5}{0}$, 6, 6]	$C^{(67)}$ [1, 4, 5, 5]
–	$P(16) = 59$	$P(17) = 61$	–	–	$P(18) = 67$

$i=69$	$i=71$	$i=73$	$i=75$	$i=77$	$i=79$
$C^{(69)}$ [$\underset{\to 3}{0}$, 3, 4, 4]	$C^{(71)}$ [2, 2, 3, 3]	$C^{(73)}$ [1,1,2, 2]	$C^{(75)}$ [$\underset{\to 3}{0}$, $\underset{\to 5}{0}$, 1, 1]	$C^{(77)}$ [2, 4, $\underset{\to 7}{0}$, $\underset{\to 11}{0}$]	$C^{(79)}$ [1, 3, 6, 10]
–	$P(19) = 71$	$P(20) = 73$	–	–	$P(21) = 79$

$i=81$	$i=83$	$i=85$	$i=87$	$i=89$	$i=91$
$C^{(81)}$ [$\underset{\to 3}{0}$, 2, 5, 9]	$C^{(83)}$ [2, 1, 4, 8]	$C^{(85)}$ [1, $\underset{\to 5}{0}$, 3, 7]	$C^{(87)}$ [$\underset{\to 3}{0}$, 4, 2, 6]	$C^{(89)}$ [2, 3, 1, 5]	$C^{(91)}$ [1, 2, $\underset{\to 7}{0}$, 4]
–	$P(22) = 83$	–	–	$P(23) = 89$	–

$i=93$	$i=95$	$i=97$	$i=99$
$C^{(93)}$ [$\underset{\to 3}{0}$, 1, 6, 3]	$C^{(95)}$ [2, $\underset{\to 5}{0}$, 5, 2]	$C^{(97)}$ [1, 4, 4, 1]	$C^{(99)}$ [$\underset{\to 3}{0}$, 3, 3, $\underset{\to 11}{0}$]
–	–	$P(24) = 97$	–

Figure 2

Algorithm steps up to 100. The scheme shows the computing of $P(\cdot)$ and $C^{(i)}(\cdot)$ for any index $i < 100$. The right arrows under $C^{(i)}(\cdot)$ indicate the assignment done (see (12) and Remark 3.10) when a prime is discovered.

As a consequence of formula (12), our algorithm reduces operations only to subtractions and assignments. Figure 2 shows the behaviour of our algorithm for the first primes up to 100.

Remark 3.13: Incidentally, we note that our algorithm is very suitable for a parallel implementation. Indeed, each processing element, connected by a network, may compute (without any conflict) one or more $C^{(i)}$, and give a prime when $C^{(i)}(j) \neq 0$ for any $j \leq l$. This theme will be extensively treated in forthcoming research.

Remark 3.14: To compute the time and the space complexity of our algorithm, we consider the usual "Big-O" notation $\mathcal{O}(\cdot)$, which refers to the worst case. This is a standard way to estimate an algorithm complexity which allows to quickly make comparisons with others. As it is well known in computers literature, searching for more accurate complexity, well as the average case complexity $\Theta(\cdot)$, tends to be very difficult. Despite its simplicity, the worst case complexity mimics well the behaviour of the asymptotic behaviour, i.e., when the size of the input goes to infinity. However, even in the worst case, a rigorous derivation of the complexity needs sophisticated techniques, involving the Turing machine. For these reasons, generally, many years occur before an exact complexity analysis. For instance, the proof that "the complexity of computing the table of primes between 1 and n on a multi tape Turing machine is $\mathcal{O}(n\ln(n))$" appears just recently (see [11]).

Hence, to compute an upper bound of our algorithm complexity, we just have to count the number of subtractions and assignments (12) done at each iteration. The algorithm requires at most $\frac{n}{2} \cdot \pi(\sqrt{n}) = \frac{n^{1.5}}{ln(n)}$ subtractions, where $\pi(\sqrt{n}) \simeq \frac{\sqrt{n}}{ln(\sqrt{n})}$ is the number of primes below $\sqrt{n}$. Moreover, the number of assignments is $nln(n)$, similarly to the complexity of sieve of Eratosthenes. Whence, the complexity of our algorithm is $\mathcal{O}\left(\frac{n^{1.5}}{ln(n)}\right)$.

In particular, the memory requirement is only $\mathcal{O}\left(\frac{n}{ln(n)}\right)$ since the sizes of P and C are $\pi(n)$ and $\pi(\sqrt{n})$, respectively.

3.3 *Prime factorization*

Notice that our algorithm can be considered also as a factorization algorithm. Indeed, according to Definition 3.5, we update the size of $C^{(i)}$

at iteration $i = P(l)^2$, where l is the size of $C^{(i-2)}$. On the contrary, if we use Definition 3.1, $C^{(i)}$ is updated as soon as a new prime is discovered. More specifically, fixed $i \leq n$, we have to compute $C^{(i)}(j)$ and, if $C^{(i)}(j) = 0$ for any $j \leq m$, then $P(j)$ is a prime factoring i. For a better understanding a reader can see Figure 3 which displays the list $C^{(i)}$, for $3 \leq i \leq 39$. As highlighted, the numbers coloured in red represent the new primes detected, while those coloured in blue refer to the primes factoring i. For example, the primes dividing $i = 39$ are the first and the fifth elements of the array $C^{(39)}$, respectively, i.e. 3 and 13. The correctness of this procedure is guaranteed by the Corollary 3.4.

$C^{(3)} = [3]$ $C^{(17)} = [2,4,2,8,11,17]$ $C^{(31)} = [1,2,2,1,4,10,13,19,28,31]$
$C^{(5)} = [2,5]$ $C^{(19)} = [1,3,1,7,10,16,19]$ $C^{(33)} = [0,1,1,0,2,9,12,18,27,30]$
$C^{(7)} = [1,4,7]$ $C^{(21)} = [0,2,0,6,9,15,18]$ $C^{(35)} = [2,0,0,10,2,8,11,17,26,29]$
$C^{(9)} = [0,3,6]$ $C^{(23)} = [2,1,6,5,8,14,17,23]$ $C^{(37)} = [1,4,6,9,1,7,10,16,25,28,37]$
$C^{(11)} = [2,2,5,11]$ $C^{(25)} = [1,0,5,4,7,13,16,22]$ $C^{(39)} = [0,3,5,8,0,6,9,15,24,27,36]$
$C^{(13)} = [1,1,4,10,13]$ $C^{(27)} = [0,4,4,3,6,12,15,21]$ ....
$C^{(15)} = [0,0,3,9,12]$ $C^{(29)} = [2,3,3,2,5,11,14,20,29]$

Figure 3

Example of prime factorization of the first 39 integers. The red numbers represent the new primes detected and the blue numbers refer to the factorization desired

4. Time optimization (stop criterion)

To improve the time realized by the suggested algorithm, we observe that as soon as $C^{(i)}(\cdot)$ becomes zero, we can stop and conclude that $i \notin \mathcal{P}$. To state this, we need to introduce the array $S^{(i)}$ which counts the number of subtractions "omitted" from the previous iteration. More formally, initialize $S^{(3)} = 0$ and define $S^{(i)}$ $(i > 3)$ as follows.

Definition 4.1 : For any $i \in \mathbb{D} - \{3\}$,

(1) if there exists j^* such that

$$j^* = \min\{1 \leq j \leq l \,|\, C^{(i)}(j^*) = 0\}, \tag{13}$$

with l given by (5), let us set

$$C^{(i)}(j) = C^{(i-2)}(j) \qquad j^* + 1 \leq j \leq l,$$

and define

$$S^{(i)}(j) = \begin{cases} 0 & 1 \leq j \leq j^* \\ S^{(i-2)}(j) + 1 & j^* + 1 \leq j \leq l, \end{cases} \tag{14}$$

(ii) otherwise, if does not exist such j^* (i.e., $i \in \mathcal{P}$, or $i = P^2(l)$), set $S^{(i)}(j) = 0$ for any $1 \le j \le l$.

From (*i*) of Definition 14, it is clear that if there exists j^* we stop to compute the next $C^{(i)}(j)$, $j > j^*$. As said above, to keep track of the operations omitted we use $S^{(i)}$, according to the next proposition.

Proposition 4.2 : For any $i \in \mathbb{D} - \{3\}$, assume there exists the index j^* defined in (13) then, for any $1 \le j \le l$, one has

$$C^{(i)}(j) = [C^{(i-2)}(j) - 1 - S^{(i-2)}(j)]_{P(j)}.$$

Proof: Consider the non trivial case in which there exists j^* such that $C^{(i-2)}(j^*) = 0$ and assume $C^{(i-2)}(j) = C^{(i-2-b)}(j)$, $b \ge 2$, for any $j^* + 1 \le j \le l$, that means there are *b* steps without updating such index *j*. Due to Proposition 3.9, we know that

$$C^{(i)}(j) = [C^{(i-2)}(j) - 1]_{P(j)} = [C^{(i-2-b)}(j) - 1 - b]_{P(j)};$$

therefore, from Definition 14, we may write,

$$C^{(i)}(j) = \begin{cases} [C^{(i-2)}(j) - 1]_{P(j)} & 1 \le j \le j^* \\ [C^{(i-2-b)}(j) - 1 - b]_{P(j)} = [C^{(i-2)}(j) - 1 - S^{(i-2)}(j)]_{P(j)} & j^* + 1 \le j \le l. \end{cases}$$

□

Proposition 3.6 still holds true to assess if the current i is prime or not. The next example shall clarify the procedure.

Example 4.3 : With refer to Figure 2, consider $i \in [31, 37]$. Since 31 and 37 are primes we have $S^{(31)}(j) = S^{(37)}(j) = 0$ for any $1 \le j \le 3$. Then, from Definition 14 and Proposition 4.2, we may write

$$C^{(31)} = [1, 2, 2], \qquad S^{(31)} = [0, 0, 0],$$

$$C^{(33)} = [\underbrace{0}_{\to 3}, 2, 2], \qquad S^{(33)} = [0, 1, 1], \qquad (j^* = 1)$$

Analogously, when $i = 35$ we have

$$C^{(35)} = [2, \underbrace{0}_{\to 5}, 2], \qquad S^{(35)} = [0, 0, 2], \qquad (j^* = 2)$$

$$C^{(37)} = [1, 4, 6], \qquad S^{(37)} = [0, 0, 0].$$

It remains clear that $S^{(i)}$ is null if $i \in \mathcal{P}$, since does not exist j^* (i.e., we never stop to update $C^{(i)}$), and if $i = P^2(l)$, since the stop occurs for the last element of $C^{(i)}$.

The advantages of this method consist of

- Minor memory accesses (memory read) on $C^{(i)}$,
- To work with an array whose size is fixed (i.e., equal to the size of $C^{(i)}$),

$i=3$	$i=5$	$i=7$	$i=9$	$i=11$	$i=13$	$i=15$	$i=17$
$C^{(3)}$ [3]	$C^{(5)}$ [2]	$C^{(7)}$ [1]	$C^{(9)}$ [0,3] ($\to 3$)	$C^{(11)}$ [2,2]	$C^{(13)}$ [1,1]	$C^{(15)}$ [0,1] ($\to 3$)	$C^{(17)}$ [2,4]
$S^{(3)}$ [0]	$S^{(5)}$ [0]	$S^{(7)}$ [0]	$S^{(9)}$ [0,0]	$S^{(11)}$ [0,0]	$S^{(13)}$ [0,0]	$S^{(15)}$ [0,1]	$S^{(17)}$ [0,0]
$i=19$	$i=21$	$i=23$	$i=25$	$i=27$	$i=29$	$i=31$	
$C^{(19)}$ [1,3]	$C^{(21)}$ [0,3] ($\to 3$)	$C^{(23)}$ [2,1]	$C^{(25)}$ [1,0,5] ($\to 5$)	$C^{(27)}$ [0,5,5] ($\to 3$)	$C^{(29)}$ [2,3,3]	$C^{(31)}$ [1,2,2]	
$S^{(19)}$ [0,0]	$S^{(21)}$ [0,1]	$S^{(23)}$ [0,0]	$S^{(25)}$ [0,0,0]	$S^{(27)}$ [0,1,1]	$S^{(29)}$ [0,0,0]	$S^{(31)}$ [0,0,0]	
$i=33$	$i=35$	$i=37$	$i=39$	$i=41$	$i=43$		
$C^{(33)}$ [0,2,2] ($\to 3$)	$C^{(35)}$ [2,0,2] ($\to 5$)	$C^{(37)}$ [1,4,6]	$C^{(39)}$ [0,4,6] ($\to 3$)	$C^{(41)}$ [2,2,4]	$C^{(43)}$ [1,1,3]		
$S^{(33)}$ [0,1,1]	$S^{(35)}$ [0,0,2]	$S^{(37)}$ [0,0,0]	$S^{(39)}$ [0,1,1]	$S^{(41)}$ [0,0,0]	$S^{(43)}$ [0,0,0]		
$i=45$	$i=47$	$i=49$	$i=51$	$i=53$	$i=55$		
$C^{(45)}$ [0,1,3] ($\to 3$)	$C^{(47)}$ [2,4,1]	$C^{(49)}$ [1,3,0,3] ($\to 7$)	$C^{(51)}$ [0,3,7,3] ($\to 3$)	$C^{(53)}$ [2,1,5,1]	$C^{(55)}$ [1,0,5,1] ($\to 5$)		
$S^{(45)}$ [0,1,1]	$S^{(47)}$ [0,0,0]	$S^{(49)}$ [0,0,0,0]	$S^{(51)}$ [0,1,1,1]	$S^{(53)}$ [0,0,0,0]	$S^{(55)}$ [0,0,1,1]		
$i=57$	$i=59$	$i=61$	$i=63$	$i=65$	$i=67$		
$C^{(57)}$ [0,5,5,1] ($\to 3$)	$C^{(59)}$ [2,3,2,9]	$C^{(61)}$ [1,2,1,8]	$C^{(63)}$ [0,2,1,8] ($\to 3$)	$C^{(65)}$ [2,0,1,8] ($\to 5$)	$C^{(67)}$ [1,4,5,5]		
$S^{(57)}$ [0,1,2,2]	$S^{(59)}$ [0,0,0,0]	$S^{(61)}$ [0,0,0,0]	$S^{(63)}$ [0,1,1,1]	$S^{(65)}$ [0,0,2,2]	$S^{(67)}$ [0,0,0,0]		
$i=69$	$i=71$	$i=73$	$i=75$	$i=77$	$i=79$		
$C^{(69)}$ [0,4,5,5] ($\to 3$)	$C^{(71)}$ [2,2,3,3]	$C^{(73)}$ [1,1,2,2]	$C^{(75)}$ [0,1,2,2] ($\to 3$)	$C^{(77)}$ [2,4,0,2] ($\to 7$)	$C^{(79)}$ [1,3,6,10]		
$S^{(69)}$ [0,1,1,1]	$S^{(71)}$ [0,0,0,0]	$S^{(73)}$ [0,0,0,0]	$S^{(75)}$ [0,1,1,1]	$S^{(77)}$ [0,0,0,2]	$S^{(79)}$ [0,0,0,0]		
$i=81$	$i=83$	$i=85$	$i=87$	$i=89$	$i=91$		
$C^{(81)}$ [0,3,6,10] ($\to 3$)	$C^{(83)}$ [2,1,4,8]	$C^{(85)}$ [1,0,4,8] ($\to 5$)	$C^{(87)}$ [0,5,4,8] ($\to 3$)	$C^{(89)}$ [2,3,1,5]	$C^{(91)}$ [1,2,0,5] ($\to 7$)		
$S^{(81)}$ [0,1,1,1]	$S^{(83)}$ [0,0,0,0]	$S^{(85)}$ [0,0,1,1]	$S^{(87)}$ [0,1,2,2]	$S^{(89)}$ [0,0,0,0]	$S^{(91)}$ [0,0,0,1]		
$i=93$	$i=95$	$i=97$	$i=99$				
$C^{(93)}$ [0,2,7,5] ($\to 3$)	$C^{(95)}$ [2,0,7,5] ($\to 5$)	$C^{(97)}$ [1,4,4,1]	$C^{(99)}$ [0,4,4,1] ($\to 3$)				
$S^{(93)}$ [0,1,1,2]	$S^{(95)}$ [0,0,2,3]	$S^{(97)}$ [0,0,0,0]	$S^{(99)}$ [0,1,1,1]				

Figure 4

The scheme shows the computing of $C^{(i)}(\cdot)$ and $S^{(i)}(\cdot)$ for any index $i < 100$

- $S^{(i)}(j)$ are globally less than $C^{(i)}(j)$, for any j. For a better understanding, the reader can see Figure 4 for the computation of $C^{(i)}$ and $S^{(i)}$, with $i < 100$.

The scheme for this extension is given below.

1. Initialize $i = 3,\ j = 1,\ P(j) = 3,\ C^{(i)}(j) = 3,$ and $S^{(i)}(j) = 0.$
2. While $i < n,$ increment $i = i + 2.$
   - While $j \le l$ (where l is the size of $C^{(i-2)}$), compute
   $$C^{(i)}(j) = [C^{(i-2)}(j) - 1]_{P(j)} \qquad 1 \le j \le l.$$
   - If there exists j^* such that
   $$j^* = \min\{1 \le j \le l \mid C^{(i)}(j^*) = 0\},$$
   then $i \notin \mathcal{P},$ and let
   $$C^{(i)}(j) = [C^{(i-2)}(j) - 1 - S^{(i-2)}(j)]_{P(j)} \qquad j^* + 1 \le j \le l.$$
   and
   $$S^{(i)}(j) = \begin{cases} 0 & 1 \le j \le j^* \\ S^{(i-2)}(j) + 1 & j^* + 1 \le j \le l, \end{cases}$$
   In particular, if $j^* = l,$ then the current index i is the square of $P(l)$,
   $$C^{(i)}(l+1) = P(l+1) - \frac{(P(l+1) - P(l))^2}{2},$$
   and $S^{(i)}$ reduces to the null array.
   - Else, if $C^{(i)}(j) \neq 0,$ for any $j \le l,$ then the index $i \in \mathcal{P},$ and update P ($S^{(i)}$ coincides with the null array).

5. Jump Version

In this section, we generalize the proposed algorithm. As explained in Section 3, we examine just the odd number to find the next prime, i.e., an even number is disregarded. An interesting improvement should consist of "jumping" across the next prime, directly. What we call *jump* between two primes represents their half-gap.

Proposition 5.1 : Let $P(L)$ be the last prime known at iteration i (where L is the number of primes discovered up to i), the jump across $P(L+1)$ belongs to the set

$$\mathcal{J}^{(i)} = \{h \in \mathbb{N}^* \mid [h]_{P(j)} \neq C^{(i)}(j), 1 \leq j \leq l\}, \tag{15}$$

which represents the set of all "admissible" jumps. Among all $h \in \mathcal{J}^{(i)}$, the "true" jump is

$$h^* = \min\{h \in \mathcal{J}^{(i)}\}.$$

Proof: By using Proposition 3.9, it is easy to verify that

$$[h]_{P(j)} \neq C^{(i)}(j) \Leftarrow \begin{cases} C^{(i+2h)}(j) \neq 0 \\ C^{(i+2[h]_{P(j)})}(j) \neq 0 \end{cases} \Leftarrow \begin{cases} P(j) \nmid (i+2h) \\ P(j) \nmid (i+2[h]_{P(j)}), \end{cases}$$

for any $1 \leq j \leq l$; this makes h a possible jump. Moreover, since

$$P(L+1) = \min\{p \in \mathbb{D} \mid i < p \wedge P(j) \nmid p, 1 \leq j \leq L\},$$

then h^* has to minimize $\mathcal{J}^{(i)}$. □

Once determined h^*, due to Proposition 3.9, we update $C^{(i+2h^*)}(j)$ as follows

$$C^{(i+2h^*)}(j) = [\mid C^{(i)}(j) - h^* \mid]_{P(j)}, \tag{16}$$

for any j; and repeat the procedure starting from $\mathcal{J}^{(i+2h^*)}$. However, we have to keep a record of the (possible) square of prime which comes before $i+2h^*$. This case can be treated according to the next proposition.

Proposition 5.2 : Given $i \in \mathbb{D} - \{3\}$, if there exists $k \in N^* - \mathcal{J}^{(i)}$ such that

$$\begin{cases} C^{(i)}(j) \neq [k]_{P(j)} & 1 \leq j < l, \\ C^{(i)}(l) = k, \end{cases} \tag{17}$$

then $(i+2k) = P^2(l)$. In this case, one has

$$C^{(i)}(l+1) = \left[P(l+1) - \frac{d^2}{2} + k \right]_{P(l+1)}. \tag{18}$$

Proof: If the condition (17) holds true we have

$$\begin{cases} C^{(i+2k)}(j) \neq 0 & 1 \leq j < l, \\ C^{(i+2k)}(l) = 0, \end{cases}$$

hence, Proposition 3.11 implies that $(i+2k) = P^2(l)$. Moreover, from Proposition 3.9 and 3.12, obtain

$$C^{(i)}(l+1) = [C^{(i+2k)} + k]_{P(l+1)} = \left[P(l+1) - \frac{d^2}{2} + k \right]_{P(l+1)}.$$

□

Finally, observe that the basic version of the suggested algorithm, presented in Section 3, is a particular case of the above jump version, in which the jump is constant and equal to 2. For a better understanding, a reader can see Example 5.3 and Figure 5.

Example 5.3 : Consider $i = 113$, for which

$$C^{(113)} = [2, 1, 3, 4].$$

The first admissible jump $h \in \mathcal{J}^{(113)}$ is $h = 5$, but $[5]_3 = 2$. The second possible h is equal to 6, but $[6]_5 = 1$. The third it is $h = 7$, which is the smallest $h \in \mathcal{J}^{(113)}$ such that $[7]_{P(j)} \neq C^{(113)}(j)$ for any $1 \leq j \leq 4$, i.e., $h^* = 7$. Moreover, note that there exists $k = 4$ satisfying

$$\begin{cases} C^{(113)}(j) \neq [k]_{P(j)} & 1 \leq j \leq 3, \\ C^{(113)}(4) = k. \end{cases}$$

Hence, we obtain that $(i+2k) = 121 (= P^2(4))$ and, from (18), $C^{(113)}(5) = [13 - (13-11)^2 / 2 + 4]_{13} = 2$.

The scheme for this version is given below.

1. Initialize $i = 3$, $j = 1$, $P(j) = 3$, and $C^{(i)}(j) = 3$.
2. While $i < n$, set $h = 1$.
   - If there exists $j (1 \leq j < l$, where l is the size of $C^{(i)})$ such that
   $$[h]_{P(j)} = C^{(i)}(j),$$
   then $(i+2h) \notin \mathcal{P}$. In particular, if
   $$\begin{cases} C^{(i)}(j) \neq [h]_{P(j)} & 1 \leq j < l, \\ C^{(i)}(l) = h, \end{cases}$$
   we get $i + 2h = P^2(L)$ (where L is the size of P), and set $C^{(i)}(l+1)$ equal to (18). Increment $h = h + 1$.

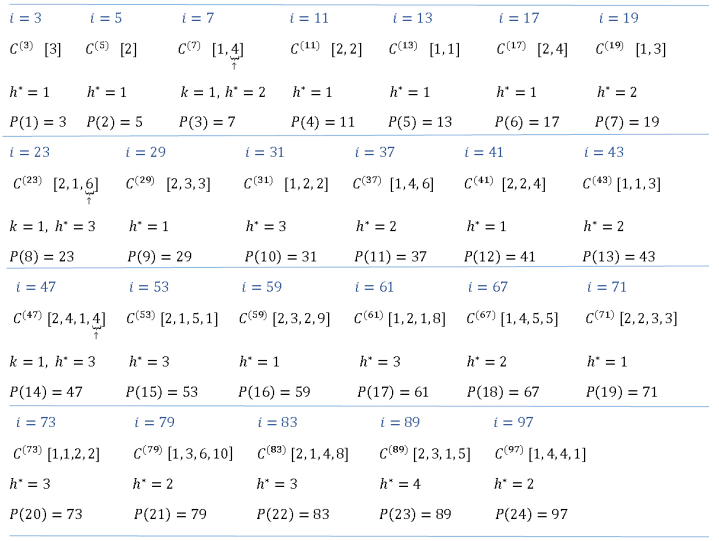

Figure 5

The scheme shows the behaviour of the "jump version" procedure, for any prime up to 100. The indices h^* and k denote the "true" jump and the half-distance across the square of the previous prime, respectively

- Else, if $[h]_{P(j)} \neq C^{(i)}(j)$, for any $1 \leq j \leq l$, set $P(L+1) = i + 2h$, i.e. $(i + 2h) \in \mathcal{P}$. Increment $i = i + 2h$ and compute $C^{(i+2h)}$ by (16).

6. Numerical Implementation

In this Section, we provide some numerical tests in which we compare our algorithm and its extension (see Sections 3-5) with the sieve of Eratosthenes, the segmented sieve of Eratosthenes and the sieve of Aitkins, respectively, taken as benchmarks. In Tables 1 and 2, we report the values of runtime and the bits required for each algorithm. In particular, the results are taken as the mean of 30 iterations in order to reduce sensitive errors. Figures 6 and 7 show the log-plot of the mentioned values. As the simulations highlight, we obtain better results about the memory and visible improvements in runtime execution for the jump version. The numerical tests are performed in Java JRE 10 with I7-4800HQ 2.6GHZ octa-core. Due to the memory limitation of the sieve of Eratosthenes we consider $n = 10^8$ as the upper bound (in particular, the upper bound for our algorithm is given by $n = 2.05 \cdot 10^9$).

Table 1

Time (ms) comparison.

n	Sieve of Eratosthenes	Segmented Sieve of Eratosthenes	Sieve of Atkins	Our Algorithm	Time Optimization	Jump Version
10^2	3.5000	3.5000	3.4667	3.5000	3.1000	2.9150
5.10^2	3.6000	3.6000	3.5333	4.0000	3.2000	2.9950
10^3	4.2000	3.7666	3.6630	4.5000	3.3000	3.0150
5.10^3	4.5000	3.8333	3.7000	6.5000	3.7000	3.5700
10^4	4.8000	4.0333	3.7667	8.5000	4.0650	4.0000
5.10^4	9.0000	6.2333	5.0667	22.0000	10.0300	5.0000
10^5	10.0000	8.5000	6.4000	49.0000	22.7650	9.0000
5.10^5	59.5000	43.9333	25.5000	337.5000	167.9650	72.0000
10^6	$1.13.10^2$	80.2666	53.6667	$8.06.10^2$	$4.2.10^2$	$1.31.10^2$
5.10^6	$0.7375.10^3$	$0.4817.10^3$	$0.4792.10^3$	$8.2355.10^3$	$3.5657.10^3$	$1.2930.10^3$
10^7	$0.1953.10^4$	$0.1114.10^4$	$0.1145.10^4$	$2.3257.10^4$	$0.8694.10^4$	$0.3319.10^4$
5.10^7	$0.1274.10^5$	$0.1267.10^5$	$0.0677.10^5$	$2.5716.10^5$	$0.8502.10^5$	$0.2601.10^5$
10^8	$0.3056.10^5$	$0.3042.10^5$	$0.1414.10^5$	$6.9231.10^5$	$1.7909.10^5$	$0.7564.10^5$

Table 2

Space (bit) comparison.

n	Sieve of Eratosthenes	Segmented Sieve of Eratosthenes	Sieve of Atkins	Our Algorithm	Time Optimization	Jump Version
10^2	$6.2537.10^3$	$5.9520.10^3$	$4.4800.10^3$	$3.5840.10^3$	$4.0320.10^3$	$3.5840.10^3$
5.10^2	$3.4637.10^5$	$3.4106.10^5$	$3.3811.10^5$	$3.3504.10^5$	$3.3523.10^5$	$3.3517.10^5$
10^3	$3.7171.10^5$	$3.6294.10^5$	$3.5411.10^5$	$3.4886.10^5$	$3.4918.10^5$	$3.4880.10^5$
5.10^3	$5.6384.10^5$	$5.2461.10^5$	$4.8211.10^5$	$4.3174.10^5$	$4.3232.10^5$	$4.3187.10^5$
10^4	$7.9552.10^5$	$7.7562.10^5$	$6.4211.10^5$	$5.1693.10^5$	$5.1770.10^5$	$5.1706.10^5$
5.10^4	$2.5752.10^6$	$2.5288.10^6$	$1.9221.10^6$	$1.1783.10^6$	$1.1798.10^6$	$1.1786.10^6$
10^5	$4.7460.10^6$	$3.7532.10^6$	$3.5221.10^6$	$2.0007.10^6$	$2.0026.10^6$	$2.0008.10^6$
5.10^5	$2.2412.10^7$	$1.7527.10^7$	$1.7099.10^7$	$0.7166.10^7$	$0.7170.10^7$	$0.7165.10^7$
10^6	$4.3920.10^7$	$4.2323.10^7$	$3.3876.10^7$	$1.3802.10^7$	$1.4286.10^7$	$1.3802.10^7$
5.10^6	$2.1270.10^8$	$2.0178.10^8$	$1.6809.10^8$	$0.6175.10^8$	$0.6176.10^8$	$0.6175.10^8$
10^7	$4.1254.10^8$	$4.0511.10^8$	$3.2748.10^8$	$1.1900.10^8$	$1.1902.10^8$	$1.1900.10^8$
5.10^7	$1.9867.10^9$	$1.6889.10^9$	$1.6025.10^9$	$0.5188.10^9$	$0.5189.10^9$	$0.5188.10^9$
10^8	$3.9422.10^9$	$3.3731.10^9$	$3.2048.10^9$	$0.9393.10^9$	$0.9396.10^9$	$0.9393.10^9$

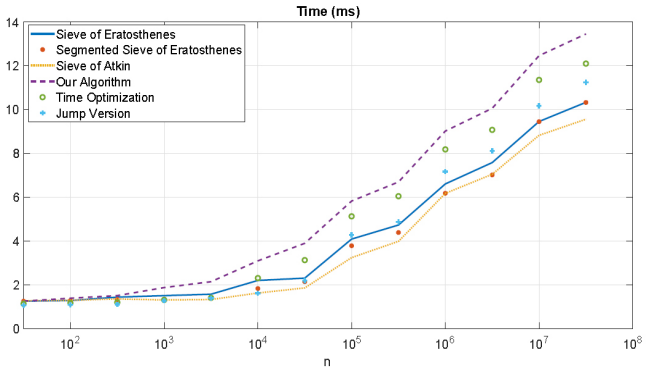

Figure 6

Log-plot of time (Table 1)

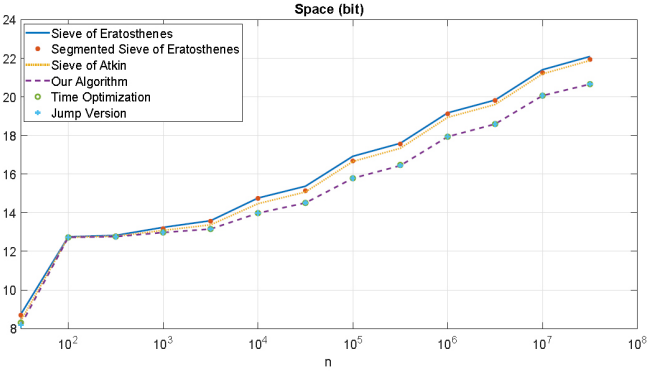

Figure 7

Log-plot of space (Table 2)

7. Conclusion

In this article we have introduced a new algorithm to find all prime numbers below a given $n \in \mathbb{N}^*$. The advantage of the suggested procedure is its "simplicity" in terms of operations performed because it uses just assignments and subtractions. As explained in Section 3, the algorithm requires less memory than other well-known sieves and is suitable for parallel implementation. This is particularly helpful when considering large numbers, as explained in Section 2. Moreover, in Sections 4 and 5 we present some extensions that increase the runtime performance while preserving the same memory requirement, as shown and tested in Section 6.

Declaration

Conflict of interest: The authors declare no conflict of interest.

Funding: The authors received no financial support for the research, authorship, and/or publication of this article.

References

[1] Abdullah, D., Rahim, R., Apdilah, D., Efendi, S., Tulus, T., and Suwilo, S. (2018). Prime numbers comparison using sieve of Eratosthenes and Sieve of Sundaram algorithm. In *Journal of Physics: Conference Series*, volume 978, page 012123. IOP Publishing.

[2] Agrawal, M., Kayal, N., and Saxena, N. (2004). PRIMES is in P. *Annals of mathematics*, pages 781–793.

[3] Atkin, A. and Bernstein, D. (2004). Prime sieves using binary quadratic forms. *Mathematics of Computation*, 73(246):1023–1030.

[4] Bombieri, E. (2000). Problems of the millennium: The Riemann hypothesis. *Clay Mathematics Institute*.

[5] Boujnouni, M. E. (2021). A study of prime numbers distribution based on support vector domain description. *Journal of Information and Optimization Sciences*, 42(4):865–882.

[6] Bufalo, M., Bufalo, D., and Orlando, G. (2021). A note on the computation of the modular inverse for cryptography. *Axioms*, 10(2):116.

[7] Hill, L. S. (1929). Cryptography in an algebraic alphabet. *The American Mathematical Monthly*, 36(6):306–312.

[8] Kahn, D. (1996). *The Codebreakers: The comprehensive history of secret communication from ancient times to the internet*. Simon and Schuster.

[9] Lehmer, D. H. (1930). An extended theory of Lucas' functions. *Annals of Mathematics*, pages 419–448.

[10] Rivest, R. L., Shamir, A., and Adleman, L. (1978). A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126.

[11] Sergeev, I. S. (2016). On the complexity of computing prime tables on a Turing machine. arXiv preprint arXiv:1604.01154.

[12] Silverman, J. H. (2014). *A friendly introduction to number theory*. Pearson.

[13] Sundarayya, P. and Vara Prasad, G. (2019). A public key cryptosystem using affine Hill cipher under modulation of prime number. *Journal of Information and Optimization Sciences*, 40(4):919–930.

[14] Trigiante, G. and Trigiante, D. (2002). A discrete approach to the prime number theorem. *The Journal of Difference Equations and Applications*, 8(1):93–100.

[15] Viswanath, M. and Kumar, M. R. (2015). A public key cryptosystem using Hill's cipher. *Journal of Discrete Mathematical Sciences and Cryptography*, 18(1-2):129–138.

[16] Wirian, D. J. (2009). Parallel prime sieve: Finding prime numbers. *Institute of Information & Mathematical Sciences Massey University at Albany*, Auckland, New Zealand.

[17] Zhang, Y. (2014). Bounded gaps between primes. *Annals of Mathematics*, pages 1121–1174.

Received September, 2021

Revised March, 2022