Algorithmic concepts for the computation of Jacobsthal's function

Mario Ziller and John F. Morack

Abstract

The Jacobsthal function has aroused interest in various contexts in the past decades. We review several algorithmic ideas for the computation of Jacobsthal's function for primorial numbers and discuss their practicability regarding computational effort. The respective function values were computed for primes up to 251. In addition to the results including previously unknown data, we provide exhaustive lists of all sequences of the appropriate maximum lengths in ancillary files.

1 Introduction

Henceforth, we denote the set of integral numbers by $\mathbb{Z}$ and the set of natural numbers, i.e. positive integers, by $\mathbb{N}$. $\mathbb{P} = \{p_i \mid i \in \mathbb{N}\}$ is the set of prime numbers with $p_1 = 2$. As usual, we define the n^{th} primorial number as the product of the first n primes: $p_n\# = \prod_{i=1}^{n} p_i\ , n \in \mathbb{N}$. We essentially follow the notation of Hagedorn 2009 [5].

The ordinary Jacobsthal function $j(n)$ is defined to be the smallest positive integer m, such that every sequence of m consecutive integers contains at least one integer coprime to n [8, 3, 5, 2].

Definition 1.1. *Jacobsthal function.* [6]
For $n \in \mathbb{N}$, the Jacobsthal function $j(n)$ is defined as

$$j(n) = \min\,\{m \in \mathbb{N} \mid \forall\, a \in \mathbb{Z}\ \exists\, q \in \{1, \ldots, m\} : a + q \perp n\}.$$

This definition is equivalent to the formulation that $j(n)$ is the greatest difference m between two terms in the sequence of integers which are coprime to n.

$$\begin{aligned} j(n) = \max\,\{m \in \mathbb{N} \mid \exists\, a \in \mathbb{Z}\ : a \perp n \wedge a + m \perp n \wedge \\ \forall\, q \in \{1, \ldots, m-1\} : a + q \not\perp n\}. \end{aligned}$$

In other words, $(j(n) - 1)$ is the greatest length $m^* = m - 1$ of a sequence of consecutive integers which are not coprime to n. For this reason, we define a reduced variant of the Jacobsthal function [8]. This will make a simplified representation possible.

Definition 1.2. *Reduced Jacobsthal function.* [13]
For $n \in \mathbb{N}$, the reduced Jacobsthal function $j^*(n)$ is defined as

$$j^*(n) = j(n) - 1 = \max\{m^* \in \mathbb{N} \mid \exists\, a \in \mathbb{Z}\, \forall\, q \in \{1, \ldots, m^*\} : a + q \not\perp n\}.$$

Remark 1. The following statements are elementary consequences of the definition of Jacobsthal's function and describe some interesting properties of it [8].

Product.
$\forall\, n1, n2 \in \mathbb{N} : j(n1 \cdot n2) \geq j(n1) \wedge j(n1 \cdot n2) \geq j(n2)$.

Coprime product.
$\forall\, n1, n2 \in \mathbb{N} > 1 \mid n1 \perp n2 : j(n1 \cdot n2) > j(n1) \wedge j(n1 \cdot n2) > j(n2)$.

Greatest common divisor.
$\forall\, n1, n2 \in \mathbb{N} : j(gcd(n1, n2)) \leq j(n1) \wedge j(gcd(n1, n2)) \leq j(n2)$.

Prime power.
$\forall\, n, k \in \mathbb{N}\, \forall\, p \in \mathbb{P} : j(p^k \cdot n) = j(p \cdot n)$.

Prime separation.
$\forall\, n, k \in \mathbb{N}\, \forall\, p \in \mathbb{P} \mid n = p^k \cdot n^*, p \perp n^* : j(n) = j(p \cdot n^*)$.

The last remark implies that the entire Jacobsthal function is determined by its values for products of distinct primes [8]. In his subsequent elaborations [8, 9, 10, 11, 12], Jacobsthal derived explicit formulae for the calculation of function values for square-free integers containing up to 7 distinct prime factors, and bounds of the function for up to 10 distinct prime factors.

The particular case of primorial numbers is therefore most interesting because the function values at these points contain the relevant information for constructing general upper bounds. The Jacobsthal function of primorial numbers $h(n)$ [5] is therefore defined as the smallest positive integer m, such that every sequence of m consecutive integers contains an integer coprime to the product of the first n primes.

Definition 1.3. *Primorial Jacobsthal function.* [7]
For $n \in \mathbb{N}$, the primorial Jacobsthal function $h(n)$ is defined as

$$h(n) = j(p_n\#).$$

By analogy with definition 1.2, we also define a reduced variant of the latter function which represents the greatest length of a sequence of consecutive integers which are not coprime to the n^{th} primorial number.

Definition 1.4. *Reduced primorial Jacobsthal function.* [15]
For $n \in \mathbb{N}$, the reduced primorial Jacobsthal function $h^*(n)$ is defined as

$$h^*(n) = h(n) - 1,$$
$$h^*(n) = \max\{m^* \in \mathbb{N} \mid \exists\, a \in \mathbb{Z}\, \forall\, q \in \{1, \ldots, m^*\} : a + q \not\perp p_n\#\}.$$

For the effective computation of $h(n)$, or $h^*(n)$ respectively, it is sufficient to omit the first prime 2 from the calculation. Therefore, we define a condensed Jacobsthal function $\omega(n)$ which is directly related to $h(n)$. Its computation reduces unnecessary effort. Hagedorn described this function in the context of killing sieves [4, 5] which we avoid here. We prefer the straightforward derivation of it for our purpose. Furthermore, we harmonise the arguments of the functions $\omega(n)$ and $h(n)$ so that n in both of them refer to the greatest considered prime p_n whereas Hagedorn [5] links $\omega(n)$ to p_{n+1}.

Definition 1.5. *Condensed Jacobsthal function.* [20]
For $n \in \mathbb{N}$, the condensed Jacobsthal function $\omega(n)$ is defined as the greatest length of a sequence of consecutive integers which are not coprime to the product of the odd primes through p_n.

$$\omega(n) = j^*(p_n\#/2).$$

In other words, $\omega(n)$ is the greatest length of a sequence of consecutive integers each of which is divisible by one of the odd primes through p_n.

The prime 2 plays a specific role for the Jacobsthal function. The following lemma describes an interesting property of it and makes the direct calculation of $j(n)$ as a function of $\omega(n)$ possible.

Lemma 1.1. *Let $n \in \mathbb{N}$ with $2 \nmid n$. Then*

$$j(2 \cdot n) = 2 \cdot j(n).$$

Proof. According to definition 1.2, we resume

$$j(n) = \max \{m \in \mathbb{N} \mid \exists\, a \in \mathbb{Z}\; \forall\, q \in \{1, \dots, m-1\} : a + q \not\perp n\}.$$

Given $m \in \mathbb{N}$ and $a \in \mathbb{Z}$ as above. Then, there exists an $a^* \in \mathbb{Z}$ with $a^* \equiv 2 \cdot a \ (mod\ n)$ and $a^* \equiv 1 \ (mod\ 2)$ due to the Chinese remainder theorem. With this a^*, we get for $q = 1, \dots, m-1$

$$a^* + 2 \cdot q \equiv 2 \cdot a + 2 \cdot q \equiv 2 \cdot (a + q) \ (mod\ n), \text{ and}$$

$$a^* + 2 \cdot q - 1 \equiv 1 + 2 \cdot q - 1 \equiv 0 \ (mod\ 2).$$

Because in addition $a^* + 2 \cdot m - 1 \equiv 0 \ (mod\ 2)$,

$$\exists\, a^* \in \mathbb{Z}\; \forall\, q \in \{1, \dots, 2 \cdot m - 1\} : a + q \not\perp 2 \cdot n\}$$

holds, and $j(2 \cdot n) \geq 2 \cdot m = 2 \cdot j(n)$ follows. The maximality remains retained in both directions. □

Corollary 1.2.

$$h(n) = 2 \cdot \omega(n) + 2.$$

Proof.
$\omega(n) = j^*(p_n\#/2) = j(p_n\#/2) - 1 = j(p_n\#)/2 - 1.$
$h(n) = j(p_n\#) = 2 \cdot \omega(n) + 2.$ □

The counting of $\omega(n)$ remains the central computational problem. By definition, $\omega(n)$ is the greatest length of a sequence of consecutive integers which are not coprime to all of the first $n-1$ odd primes $p_2, \dots, p_n$. This means that every integer of the sequence must be divisible by at least one of these primes. On the other hand, those sequences can be characterised by a unique choice of non-zero residue classes for each prime.

Proposition 1.3. *Let $m, n \in \mathbb{N}$, $n > 1$.*
The following two statements are equivalent.

(1) There is an $a \in \mathbb{Z}$ so that every integer of the sequence $\{a+1, \dots, a+m\}$ is divisible by one of the primes $p_2, \dots, p_n$.

$$\exists\, a \in \mathbb{Z}\ \forall\, q \in \{1, \dots, m\}\ \exists\, i \in \{2, \dots, n\} : a + q \equiv 0 \ (mod\ p_i).$$

(2) For every prime $p_2, \dots, p_n$, there exists one non-zero residue class so that every integer of the sequence $\{1, \dots, m\}$ belongs to one of them.

$$\begin{aligned} &\exists\, a_i \in \{1, \dots, p_i - 1\},\ i = 2, \dots, n \\ &\forall\, q \in \{1, \dots, m\}\ \exists\, i \in \{2, \dots, n\} : q \equiv a_i \ (mod\ p_i). \end{aligned}$$

Proof.
(1) $\Rightarrow$ (2):
$a_i \equiv -a \ (mod\ p_i),\ i = 2, \dots, n$ satisfy the respective congruences of (2).
(2) $\Rightarrow$ (1):
According to the Chinese remainder theorem, there exists an $a \in \mathbb{Z}$ solving the system of simultaneous congruences $a \equiv -a_i \ (mod\ p_i),\ i = 2, \dots, n$. With this solution a, $\{a+1, \dots, a+m\}$ fulfils (1). □

Remark 2. In this proposition, m is not necessarily the maximum sequence length. It remains true for $m \leq \omega(n)$, too. Furthermore, the proposition holds for any set of distinct primes. The specific choice of the primes was not used in the proof.

Corollary 1.4. *For every sequence of maximum length satisfying proposition 1.3 (2), there exists a reverse sequence with*

$$\begin{aligned} &\exists\, b_i \in \{1, \dots, p_i - 1\},\ i = 2, \dots, n \\ &\forall\, q \in \{1, \dots, m\}\ \exists\, i \in \{2, \dots, n\} : m + 1 - q \equiv b_i \ (mod\ p_i). \end{aligned}$$

Proof. The maximum length of the given sequence implies $m + 1 \not\equiv a_i \ (mod\ p_i)$ for all $i = 2, \dots, n$. Therefore, $b_i \equiv m + 1 - a_i \ (mod\ p_i),\ i = 2, \dots, n$ satisfy the requirements. □

The pairs of reverse sequences define a symmetry within the set of sequences of maximum length. The algorithmic exploitation of this interesting feature, however, seems to be difficult because m is a priori unknown.

There is another way to characterise a sequence with the considered properties. Given a permutation of the primes $\{p_2, \dots, p_n\}$, the sequence can be constructed from left to right covering the next free position recursively. Conversely, a permutation of this kind can be derived from a given sequence.

Proposition 1.5. *Let $m, n \in \mathbb{N}$, $n > 1$.*
The following two statements are equivalent if $m = \omega(n)$.

(2) *For every prime $p_2, \ldots, p_n$, there exists one non-zero residue class so that every integer of the sequence $\{1, \ldots, m\}$ belongs to one of them.*

$$\exists\, a_i \in \{1, \ldots, p_i - 1\},\ i = 2, \ldots, n$$
$$\forall\, q \in \{1, \ldots, m\}\ \exists\, i \in \{2, \ldots, n\} : q \equiv a_i\ (mod\ p_i).$$

(3) *There exists a permutation $(\pi_2, \ldots, \pi_n)$ of $\{p_2, \ldots, p_n\}$ and a tuple $(q_2, \ldots, q_n)$ so that the sequence $\{1, \ldots, m\}$ is completely covered by the residue classes q_i mod π_i when all π_i were recursively assigned to the first free position q_i, respectively.*

$$\begin{aligned}
&\exists\, \pi_i \in \{p_2, \ldots, p_n\}\ \exists\, q_i \in \{1, \ldots, m\},\ i = 2, \ldots, n \\
&\quad \text{with } \{\pi_2, \ldots, \pi_n\} \setminus \{p_2, \ldots, p_n\} = \emptyset, \\
&\quad q_i \not\equiv 0\ (mod\ \pi_i),\ i = 2, \ldots, n, \\
&\quad i < j \Rightarrow q_i < q_j,\ i, j \in \{2, \ldots, n\}, \\
&\quad q_2 = 1, \text{ and} \\
&\quad q_i = min\{j \in \{1, \ldots, m\} \mid \forall k < i : j \not\equiv q_k\ (mod\ \pi_k)\}, i = 3, \ldots, n \\
&\forall\, q \in \{1, \ldots, m\}\ \exists\, i \in \{2, \ldots, n\} : q \equiv q_i\ (mod\ \pi_i).
\end{aligned}$$

Proof.
(2) $\Rightarrow$ (3):
We set $q_2 = 1$ and choose the smallest p_j with $a_j \equiv 1\ (mod\ p_j)$ as π_2. Given q_k and π_k for $2 \le k < i \le n$, we set $q_i = min\{j \in \{1, \ldots, m\} \mid \forall k < i : j \not\equiv q_k\ (mod\ \pi_k)\}$. Then we choose π_i as the smallest p_j with $a_j \equiv q_i\ (mod\ p_j)$. There must exist a proper p_j because $m = \omega(n)$. Otherwise, at least p_j was not needed to cover a sequence of length m and this sequence could be extended to a longer one by setting $q_i = m + 1$ and $\pi_i = p_j$ which contradicts to the definition of $\omega(n)$. All q_i and π_i inductively selected as described above fulfils (3).
(3) $\Rightarrow$ (2):
For every $i \in \{2, \ldots, n\}$, there exists a unique $j \in \{2, \ldots, n\}$ with $\pi_j = p_i$ because $(\pi_2, \ldots, \pi_n)$ is a permutation of $\{p_2, \ldots, p_n\}$. $a_i \equiv q_j\ (mod\ p_i)$, $i = 2, \ldots, n$ satisfies the respective congruences of (2). □

Remark 3. The set of sequences with the properties (2) include any sequence fulfilling (3), even if $m < \omega(n)$. The proof did not make use of that condition. The proposition again holds for any set of distinct primes. The specific choice of the prime set was also not used in the proof.

Example 1. We give an example for $n = 6$ for all of the three variants of the propositions 1.3 and 1.5. The primes to be considered are 3, 5, 7, 11, and 13. The result $\omega(n)$=10 is reached in a sequence with the following properties:
Prime sequence. Proposition 1.3 (1).
$a = 12227$.
$3/(a+1)$, $7/(a+2)$, $5/(a+3)$, $3/(a+4)$, $11/(a+5)$,

$13/(a+6),\ 3/(a+7),\ 5/(a+8),\ 7/(a+9),\ 3/(a+10)$.

Set of remainders. Proposition 1.3 (2) or proposition 1.5 (2).

$a_2 = 1,\ a_3 = 3,\ a_4 = 2,\ a_5 = 5,\ a_6 = 6$.

$1 \equiv 1\ (mod\, 3),\ 2 \equiv 2\ (mod\, 7),\ 3 \equiv 3\ (mod\, 5),\ 4 \equiv 1\ (mod\, 3),\ 5 \equiv 5\ (mod\, 11),$
$6 \equiv 6\ (mod\, 13),\ 7 \equiv 1\ (mod\, 3),\ 8 \equiv 3\ (mod\, 5),\ 9 \equiv 2\ (mod\, 7),\ 10 \equiv 1\ (mod\, 3)$.

Prime Permutation. Proposition 1.5 (3).

$\pi_2 = 3,\ \pi_3 = 7,\ \pi_4 = 5,\ \pi_5 = 11,\ \pi_6 = 13$.

$q_2 = 1,\ q_3 = 2,\ q_4 = 3,\ q_5 = 5,\ q_6 = 6$.

$1 \equiv 1\ (mod\, 3),\ 2 \equiv 2\ (mod\, 7),\ 3 \equiv 3\ (mod\, 5),\ 4 \equiv 1\ (mod\, 3),\ 5 \equiv 5\ (mod\, 11),$
$6 \equiv 6\ (mod\, 13),\ 7 \equiv 1\ (mod\, 3),\ 8 \equiv 3\ (mod\, 5),\ 9 \equiv 2\ (mod\, 7),\ 10 \equiv 1\ (mod\, 3)$.

Remark 4. As a consequence of the propositions 1.3 and 1.5, we can formulate three equivalent descriptions of the condensed Jacobsthal function $\omega(n)$ for $n > 1$.

(1) The function $\omega(n)$ is the maximum length m of a sequence of consecutive integers where each of them is divisible by at least one of the first n odd primes $p_2, \ldots, p_n$.

$$\omega(n) = \max \{m \in \mathbb{N} \mid \exists\, a \in \mathbb{Z}\ \ \forall\, q \in \{1, \ldots, m\}\ \exists\, i \in \{2, \ldots, n\} : a + q \equiv 0\ (mod\ p_i)\}.$$

(2) The function $\omega(n)$ is the maximum $m \in \mathbb{N}$ for which there exists a set of remainders $a_i\ mod\ p_i,\ i = 2, \ldots, n$ so that every $q \in \{1, \ldots, m\}$ satisfies one of the congruences $q \equiv a_i\ (mod\ p_i)$.

$$\omega(n) = \max \{m \in \mathbb{N} \mid \exists\, a_i \in \{1, \ldots, p_i - 1\},\ i = 2, \ldots, n\ \ \forall\, q \in \{1, \ldots, m\}\ \exists\, i \in \{2, \ldots, n\} : q \equiv a_i\ (mod\ p_i)\}.$$

(3) The function $\omega(n)$ is the maximum $m \in \mathbb{N}$ for which there exists a permutation $(\pi_2, \ldots, \pi_n)$ of $\{p_2, \ldots, p_n\}$ and a tuple $(q_2, \ldots, q_n)$ so that the sequence $\{1, \ldots, m\}$ is completely covered by the residue classes q_i mod π_i when all π_i were recursively assigned to the first free position q_i, respectively.

$$\begin{aligned}\omega(n) = \max \{m \in \mathbb{N} \mid\ & \exists\, \pi_i \in \{p_2, \ldots, p_n\}\ \exists\, q_i \in \{1, \ldots, m\},\ i = 2, \ldots, n \\ & \textit{with } \{\pi_2, \ldots, \pi_n\} \setminus \{p_2, \ldots, p_n\} = \emptyset, \\ & q_i \not\equiv 0\ (mod\ \pi_i),\ i = 2, \ldots, n, \\ & i < j \Rightarrow q_i < q_j,\ i, j \in \{2, \ldots, n\}, \\ & q_2 = 1,\ \textit{and} \\ & q_i = min\{j \in \{1, \ldots, m\} \mid \forall k < i : j \not\equiv q_k\ (mod\ \pi_k)\}, i = 3, \ldots, n \\ & \forall\, q \in \{1, \ldots, m\}\ \exists\, i \in \{2, \ldots, n\} : q \equiv q_i\ (mod\ \pi_i)\}.\end{aligned}$$

In the next chapter, we will review several algorithmic ideas for the computation of Jacobsthal's function for primorial numbers. All of these approaches utilise the versions (2) or (3) of understanding $\omega(n)$. When we present our results below, we will get back to the recent remark in order to derive demonstrative forms for the presentation of sequences.

2 Computation of $\omega(n)$

2.1 Basic algorithms

Brute force is the most obvious idea for computing $\omega(n)$. All possible remainder combinations according to proposition 1.3, statement (2), are processed sequentially. To each of them, a fill&cont procedure is applied as follows. For each of the residue classes, all positions of a previously empty array of sufficient length covered by that residue class are labelled. The first unlabelled position corresponds to $m + 1$ where m is the length of the related sequence. This length is then registered for searching the maximum possible length.

This naïve algorithm is henceforth referred to as Basic Sequential Algorithm (BSA). It can be implemented as a recursive procedure as depicted in the pseudocode 1.

Algorithm 1 Basic Sequential Algorithm (BSA).

```
procedure BASIC_SEQUENTIAL(arr,k)
    for i=1 to plist[k]-1 do
        arr1=arr; fill_array(arr1,i,plist[k])
        if k<n-1 then basic_sequential(arr1,k+1)
        else count_array(arr1)
        end if
    end for
end procedure
arr=empty_array                              ▷ Sequence array
plist=[$p_2$,...,$p_n$]                      ▷ Array of primes
k=1                                          ▷ Starting prime array index
basic_sequential(arr,k)                      ▷ Recursion
```

The filling of a previously empty array of sufficient length from left to right according to proposition 1.5, statement (3), is another simple idea for computing $\omega(n)$. All permutations $(\pi_2, \ldots, \pi_n)$ of the given primes $\{p_2, \ldots, p_n\}$ are processed sequentially. Starting with position 1, the first prime π_2 is chosen and with it the residue class 1 mod π_2. After labelling all positions covered by that residue class, the next unlabelled position is searched and the next prime of the permutation under consideration is analogously used until all primes are consumed.

If in any case a prime divides the index of the next unlabelled position then this case can be omitted because only non-zero residue classes are appropriate. Handling all permutations where primes are not divisors of their related position-numbers will therefore discover all sequences of the maximum possible length. This Basic Permutation Algorithm is described in pseudocode 2.

Algorithm 2 Basic Permutation Algorithm (BPA).

```
procedure BASIC_PERMUTATION(arr,plist,k,q)
    for i=k to n-1 do
        if q $\not\equiv 0 \pmod{\text{plist[i]}}$ then
            arr1=arr; fill_array(arr1,q,plist[i])
            if k<n-1 then
                plist1=plist; interchange(plist1[k],plist1[i])
                k1=k+1; q1=next_free_position(arr1)
                basic_permutation(arr1,plist1,k1,q1)
            else count_array(arr1)
            end if
        end if
    end for
end procedure
arr=empty_array                                   ▷ Sequence array
plist=[$p_2$,...,$p_n$]                              ▷ Array of primes
k=1                                    ▷ Starting prime array index
q=1                                   ▷ Starting sequence position
basic_permutation(arr,plist,k,q)                        ▷ Recursion
```

Both of the presented algorithms are not really effective for practical purpose. A simple estimation of their complexities reveals the problem. The number N_{BSA} of residue class combinations in the Basic Sequential Algorithm is

$$N_{BSA} = \prod_{i=2}^{n} (p_i - 1).$$

For the Basic Permutation Algorithm, we have

$$N_{BPA} \leq (n-1)! \ll N_{BSA}.$$

However, $(n-1)!$ also grows too fast to be reasonable. In the upcoming sections, we describe possibilities of how to dramatically reduce the number of sequences which are needed to be considered for determining $\omega(n)$.

The Basic Permutation Algorithm as described above is ineffective for another reason. There may exist different, equivalent permutations constituting the same sequence. In this case, only one of these permutations must be examined. Selecting the smallest prime if there are any at choice, defines the additional rule of a Reduced Permutation Algorithm (RPA) [22] which thereby also downsizes the number of permutations needed to be considered. This principle was yet applied in part (2) $\Rightarrow$ (3) of the proof of proposition 1.5. The Basic Permutation Algorithm 2 indeed produces doublets of sequences.

Proposition 2.1. *Let $m, n \in \mathbb{N}$, $n > 1$, and $m = \omega(n)$.*

Given a permutation $(\pi_2, \ldots, \pi_n)$ of $\{p_2, \ldots, p_n\}$ and a tuple $(q_2, \ldots, q_n)$ so that the sequence $\{1, \ldots, m\}$ is completely covered by the residue classes q_i mod π_i when all π_i were *recursively assigned to the first free position q_i, respectively. And let $q_{k_1} \equiv q_{k_2}$ $(mod\ \pi_{k_2})$, $k_1 < k_2$, for any $k_1, k_2 \in \{2, \dots, n\}$.*

$$\begin{aligned}
&\forall\, q \in \{1, \dots, m\}\ \exists\, i \in \{2, \dots, n\} : q \equiv q_i\ (mod\ \pi_i) \\
&\quad where\ \pi_i \in \{p_2, \dots, p_n\}, q_i \in \{1, \dots, m\},\ i = 2, \dots, n, \\
&\quad \{\pi_2, \dots, \pi_n\} \setminus \{p_2, \dots, p_n\} = \emptyset, \\
&\quad q_i \not\equiv 0\ (mod\ \pi_i),\ i = 2, \dots, n, \\
&\quad i < j \Rightarrow q_i < q_j,\ i, j \in \{2, \dots, n\}, \\
&\quad q_2 = 1,\ and \\
&\quad q_i = min\{j \in \{1, \dots, m\} \mid \forall k < i : j \not\equiv q_k\ (mod\ \pi_k)\}, i = 3, \dots, n. \\
&\exists\, k_1, k_2 \in \{2, \dots, n\}, k_1 < k_2 : q_{k_1} \equiv q_{k_2}\ (mod\ \pi_{k_2}).
\end{aligned}$$

Then there exist an equivalent permutation $(\varrho_2, \dots, \varrho_n)$ of $\{p_2, \dots, p_n\}$ and a tuple $(r_2, \dots, r_n)$ according to proposition 1.5, statement (3) so that $\varrho_k = \pi_k$, $r_k = q_k$ for $k < k_1$ if exist, and $\varrho_{k_1} = \pi_{k_2}$, $r_{k_1} = q_{k_1}$. Furthermore for $k > k_1$ if exist, $r_k \equiv q_j$ $(mod\ \varrho_k)$ if $\varrho_k = \pi_j$.

$$\begin{aligned}
&\exists\, \varrho_i \in \{p_2, \dots, p_n\}\ \exists\, r_i \in \{1, \dots, m\},\ i = 2, \dots, n \\
&\quad with\ \{\varrho_2, \dots, \varrho_n\} \setminus \{p_2, \dots, p_n\} = \emptyset, \\
&\quad r_i \not\equiv 0\ (mod\ \varrho_i),\ i = 2, \dots, n, \\
&\quad i < j \Rightarrow r_i < r_j,\ i, j \in \{2, \dots, n\}, \\
&\quad r_2 = 1, \\
&\quad r_i = min\{j \in \{1, \dots, m\} \mid \forall\, k < i : j \not\equiv r_k\ (mod\ \varrho_k)\}, i = 3, \dots, n, \\
&\quad \forall\, q \in \{1, \dots, m\}\ \exists\, i \in \{2, \dots, n\} : q \equiv r_i\ (mod\ \varrho_i). \\
&\quad \forall\, k \in \{2, \dots, n\},\ k < k_1 : \varrho_k = \pi_k \wedge\ r_k = q_k, \\
&\quad \varrho_{k_1} = \pi_{k_2},\ r_{k_1} = q_{k_1}, \\
&\forall\, k \in \{2, \dots, n\},\ k > k_1 : r_k \equiv q_j\ (mod\ \varrho_k)\ if\ \varrho_k = \pi_j.
\end{aligned}$$

Proof. According to the assumptions, r_i and ϱ_i satisfy the requirements for $i \le k_1$. Given r_k and ϱ_k for $k_1 \le k < i \le n$, we set

$$r_i = min\{j \in \{1, \dots, m\} \mid \forall\, k < i : j \not\equiv r_k\ (mod\ \varrho_k)\}.$$

There must exist one of the remaining primes $\pi_j \in \{\pi_{k_1}, \dots, \pi_n\} \setminus \{\varrho_{k_1}, \dots, \varrho_{i-1}\}$ with $r_i \equiv q_j$ $(mod\ \pi_j)$ because $\{1, \dots, m\}$ is completely covered and $m = \omega(n)$ is maximum. If there are several appropriate primes π_j to choose from, we set $\varrho_i = \pi_j$ for the smallest π_j possible. These r_i and ϱ_i inductively selected as described above complete the wanted permutation and tuple. □

Example 2. We give an example for $n = 8$. The primes to be considered are 3, 5, 7, 11, 13, 17, and 19. The result is $\omega(n)$=16.
The permutations
$\pi_2 = 3,\ \pi_3 = 13,\ \pi_4 = 11,\ \pi_5 = 7,\ \pi_6 = 5,\ \pi_7 = 17,\ \pi_8 = 17$, with
$q_2 = 1,\ q_3 = 2,\ q_4 = 3,\ q_5 = 5,\ q_6 = 6,\ q_7 = 8,\ q_8 = 9$,
and

$\varrho_2 = 5,\ \varrho_3 = 13,\ \varrho_4 = 11,\ \varrho_5 = 3,\ \varrho_6 = 7,\ \varrho_7 = 17,\ \varrho_8 = 17$, with
$r_2 = 1,\ r_3 = 2,\ r_4 = 3,\ r_5 = 4,\ r_6 = 5,\ r_7 = 8,\ r_8 = 9$
are equivalent because
$k_1 = 2,\ k_2 = 6$ with $q_{k_1} \equiv q_{k_2}\ (mod\ \pi_{k_2})$, i.e. $1 \equiv 6\ (mod\ 5)$, and
$\varrho_{k_1} = \pi_{k_2},\ r_{k_1} = q_{k_1}$.
The other required congruences are obvious.

The Reduced Permutation Algorithm (RPA) as depicted in pseudocode 3 makes use of proposition 2.1 and skips all permutations for which there exists an equivalent permutation with a smaller prime at a compatible position.

Algorithm 3 Reduced Permutation Algorithm (RPA).

```
procedure REDUCED_PERMUTATION(arr,plist,k,qlist)
    for i=k to n-1 do
        if qlist[k] $\not\equiv$ 0 (mod plist[i]) then
            if forall j<k : qlist[j]$\not\equiv$qlist[k] (mod plist[k]) or plist[j]<plist[k] then
                arr1=arr; fill_array(arr1,qlist[k],plist[i])
                if k<n-1 then
                    plist1=plist; interchange(plist1[k],plist1[i])
                    qlist[k+1]=next_free_position(arr1)
                    reduced_permutation(arr1,plist1,k+1,qlist)
                else count_array(arr1)
                end if
            end if
        end if
    end for
end procedure
arr=empty_array                                        ▷ Sequence array
plist=[$p_2$,...,$p_n$]                                 ▷ Array of primes
k=1                                       ▷ Starting prime array index
qlist=empty_array                    ▷ List of first unlabelled positions
qlist[k]=1                                ▷ Starting sequence position
reduced_permutation(arr,plist,k,qlist)                     ▷ Recursion
```

2.2 Linear programming

There is another, direct way of computing $\omega(n)$. The problem of determining the maximum length of a sequence covered by a choice of residue classes according to proposition 1.3, statement (2), can be reformulated as a linear program. This idea was recently applied by Resta to a similar topic [19]. The possibility of simply using standard software seems to be tempting. It is not an algorithmic idea itself. But we outline it in this section for the sake of completeness, simplicity, and mathematical straightforwardness.

Resuming statement (2) of proposition 1.3, there exists one non-zero residue class for every prime $p_2, \ldots, p_n$ so that every integer of the sequence $\{1, \ldots, m\}$ belongs to one of them.

$$\begin{aligned}
&\exists\, a_i \in \{1, \ldots, p_i - 1\},\ i = 2, \ldots, n \\
&\forall\, q \in \{1, \ldots, m\}\ \exists\, i \in \{2, \ldots, n\} : q \equiv a_i \ (mod\ p_i).
\end{aligned}$$

For every possible residue class, we define a binary variable $x_{i,j} \in \{0, 1\}$, $i \in \{2, \ldots, n\}, j \in \{1, ..., p_i - 1\}$ where $x_{i,j} = 1$ if and only if $j = a_i$.

For the description of the sequence characterised as above, we formulate two sets of constraints. First, exactly one remainder should be chosen for each prime. Second, every position in the sequence $1, ..., m$ should be covered by at least one of the residue classes.

$$\begin{aligned}
\sum_{j=1}^{p_i-1} x_{i,j} &= 1, \quad i = 2, \ldots, n, \\
\sum_{\substack{i=2 \\ p_i \nmid j}}^{n} x_{i,j\ mod\ p_i} &\geq 1, \quad j = 1, ..., m.
\end{aligned} \tag{2.1}$$

The question of interest is whether or not there exist feasible solutions of these constraints. An objective function is not really needed. So, a dummy objective like

$$\max \sum_{i=2}^{n} \sum_{j=1}^{p_i-1} x_{i,j}$$

completes the integer linear program (ILP) with binary variables.

In order to calculate $\omega(n)$, a trial and error approach like nested intervals or the like might be applied, solving the described ILP for different m. If the system has any feasible solutions for m and none for $m + 1$ then $\omega(n) = m$.

The search for the maximum suitable m can also be embedded into a single, extended ILP. Let $m_1 < \omega(n)$ and $m_2 > \omega(n)$ be tentative assumptions. We define additional binary variables $y_k \in \{0, 1\},\ k \in \{m_1, \ldots, m_2\}$ where $y_k = 1$ if and only if position k in the sequence is covered by any of the residue classes under consideration, and formulate the following linear program with binary variables.

$$
\begin{array}{lllr}
\max & \displaystyle\sum_{k=m_1}^{m_2} 2^{m_2-k}\cdot y_k, & & \\
\displaystyle\sum_{j=1}^{p_i-1} x_{i,j} & & = 1, \quad i = 2,\dots,n, & \\
\displaystyle\sum_{\substack{i=2\\ p_i \nmid j}}^{n} x_{i,j \bmod p_i} & & \geq 1, \quad j = 1,\dots,m_1-1, & (2.2)\\
\displaystyle\sum_{\substack{i=2\\ p_i \nmid k}}^{n} x_{i,k \bmod p_i} & - y_k & \geq 0, \quad k = m_1,\dots,m_2. &
\end{array}
$$

This linear program can be solved by using an established ILP solver. There are three kinds of solutions:

(1) $y_k = 0$ for all $k = m_1, \dots, m_2$.

The choice of m_1 was too large, $\omega(n) < m_1$. Another try with a smaller m_1 is needed.

(2) $y_k = 1$ for all $k = m_1, \dots, m_2$.

The choice of m_2 was too small, $\omega(n) \geq m_2$. Another try with a larger m_2 is needed.

(3) $y_k = 1$ for all $k = m_1, \dots, m < m_2, \quad y_{m+1} = 0$.
Then $\omega(n) = m$. An appropriate sequence of the length m can be derived from the solution for the variables $x_{i,j}$. A longer sequence with the required properties cannot exist because of the choice of the objective function. Each of its coefficients is larger than the sum of the following ones.

The above described way of calculating $\omega(n)$ is very simple but its feasibility is limited as well. The required computation time rapidly grows with the increasing dimension of the problem.

2.3 Bounding the remaining number of coprimes

The principles of the basic algorithms described above can only be reasonably utilised if the exclusion of many of the theoretically possible cases from the calculation can be conclusively substantiated. The Basic Sequential Algorithm processes all potential combinations of residue classes. When particular combinations can be excluded from the analysis, the computational effort can be improved upon as is done in the Reduced Permutation Algorithm. A remainder combination may be discarded because it actually cannot cover a sequence of the considered length. This decision must be based on proved criteria, which we derive in the following section.

The general idea was independently realised by Hagedorn [5] and Morack [16] in very similar approaches. Euler's totient function $\varphi(n)$ is known to be the number of positive integers up to a given integer n that are coprime to n. We use a specific generalisation of the totient function for the introduction of the problem and borrow its notation from Costello and Watts [1].

The function $\varphi(a,m,k)$ is the number of integers in the sequence $a+1,\dots,a+m$ which are coprime to $P_k = \prod_{i=2}^{k} p_i$. In other words, $\varphi(a,m,k)$ is the number of uncovered positions in the given sequence of length m where the set of remainders is condensed in a.

Definition 2.1. For $a \in \mathbb{Z}, m \in \mathbb{N}, k \in \mathbb{N} \geq 2$, we define

$$\begin{aligned}
&\varphi(a,m,1) = m,\\
&\varphi(a,m,k) = |\{q \in \{1,\dots,m\} \mid \forall\, i \in \{2,\dots,k\} : a+q \not\equiv 0 \ (mod\ p_i)\}|, \ and\\
&\nu(a,m,k) = \varphi(a,m,k-1) - \varphi(a,m,k).
\end{aligned}$$

The difference $\nu(a,m,k)$ corresponds to the number of integers coprime to $p_2,\dots,p_{k-1}$ which are not coprime to p_k, i.e. which are divisible by p_k.

Remark 5. The sequence is completely covered if $\varphi(a,m,n) = 0$. The successive contribution of prime p_k is expressed by $\nu(a,m,k)$. Thus for $2 \leq k \leq n$, we get

$$\omega(n) = max\{m \in \mathbb{N} \mid \exists\, a \in \mathbb{Z} : \varphi(a,m,n) = \varphi(a,m,k-1) - \sum_{i=k}^{n} \nu(a,m,i) = 0\}.$$

Let a solve the simultaneous congruences $a \equiv -a_j \ (mod\ p_j)$ for $j = 2,\dots,n$. Given a tentative length m of the sequence, the further processing of the corresponding set of remainders $\{a_j\}$ can be skipped if

$$\sum_{i=k}^{n} \nu(a,m,i) < \varphi(a,m,k-1)$$

can be proved for any k. Whereas $\varphi(a,m,k-1)$ can directly be counted, we below derive suitable upper bounds for $\sum_{i=k}^{n} \nu(a,m,i)$ for an early and effective decision whether this inequality applies.

In a first step, we simplify our estimate and make it independent on specific sequences, i.e independent on a, or a_j, respectively. We consider the lowest possible number $\varphi_{min}(m,k)$ of m consecutive integers which are coprime to all of the primes $p_2,\dots,p_k$, and the highest possible number $\nu_{max}(m,k)$ of m consecutive integers which are coprime to $p_2,\dots,p_{k-1}$ but divisible by p_k.

Definition 2.2. Let $m,k \in \mathbb{N}$. Then

$$\varphi_{min}(m,k) = \min_{a \in \mathbb{Z}} \varphi(a,m,k),$$

$$\nu_{max}(m,k) = \max_{a \in \mathbb{Z}} \nu(a,m,k), \ k \geq 2.$$

The term $\nu_{max}(m,k)$ can be bounded as the result of combinatorial considerations as follows.

Lemma 2.2. *Let $\lfloor x \rfloor$ denote the maximum integer not exceeding x, and $m \in \mathbb{N}$, $k \in \mathbb{N} \geq 2$.*

$$\nu_{max}(m,k) \leq r_{m,k} - \varphi_{min}(r_{m,k}, k-1)$$

where $r_{m,k} = 1 + \left\lfloor \frac{m-1}{p_k} \right\rfloor$.

Proof. The number of multiples of p_k in a sequence of m consecutive integers is at most $r_{m,k}$. Every sequence contains at least $r_{m,k} - 1$ such terms. Let $p_k \cdot (a+j)$, $a \in \mathbb{Z}$ and $j = 1, \ldots, r_{m,k}$, be the representation of $r_{m,k}$ consecutive multiples of p_k.

Every term $p_k \cdot (a+j)$ is divisible by p_i for $i = 2, \ldots, k-1$ if and only if $p_i \mid (a+j)$ because $p_i \perp p_k$.

$$p_i \mid (a+j) \Leftrightarrow p_i \mid p_k \cdot (a+j).$$

The sequence $\{a+1, \ldots, a+r_{m,k}\}$ contains multiples of p_i at the same positions j as the progression $p_k \cdot (a+j)$. The number of elements in this progression which are coprime to any p_i, $i = 2, \ldots, k-1$ is therefore $\varphi(a, r_{m,k}, k-1) \geq \varphi_{min}(r_{m,k}, k-1)$. □

The term $r_{m,k}$ can directly be calculated whereas the function $\varphi_{min}(m,k)$ requires complex computations. For this purpose, we applied a brute force algorithm very similar to the Basic Sequential Algorithm 1. While BSA intends to maximise the number of covered positions in a given array, the computation of $\varphi_{min}(m,k)$ needs to minimise it. We provide a table of function values of $\varphi_{min}(m,k)$ for $m \leq 500$ and $k \leq 11$ in an ancillary file.

The function $\varphi_{min}(m,k)$ grows very slowly with increasing k while the time needed for it explodes. So, it must be sufficient to limit the maximum considered prime for practical application.

Corollary 2.3. *Let $a \in \mathbb{Z}$, $m,n,x \in \mathbb{N}$, $k \in \mathbb{N} \geq 2$, and $r_{m,i} = 1 + \left\lfloor \frac{m-1}{p_i} \right\rfloor$, $i = k, \ldots, n$. For any $x < k$*

$$\sum_{i=k}^{n} \nu(a,m,i) \leq \sum_{i=k}^{n} r_{m,i} - \sum_{i=k}^{n} \varphi_{min}(r_{m,i}, x).$$

Proof. According to lemma 2.2, $\nu(a,m,i) \leq \nu_{max}(m,i) \leq r_{m,i} - \varphi_{min}(r_{m,i}, i-1)$ holds for any $i = k, \ldots, n$. The function $\varphi_{min}(m,k)$ is monotonically decreasing in k because the number of integers not coprime to the respective next greater prime cannot be negative. So, $\varphi_{min}(r_{m,i}, i-1) \geq \varphi_{min}(r_{m,i}, x)$ is a consequence of $x < k \leq i$. □

With the help of this corollary, we can efficiently improve the naïve sequential algorithm. We recall that the further processing of the set of remainders $\{a_k\}$ for a tentative sequence length m can be skipped if $\sum_{i=k}^{n} \nu(a,m,i) < \varphi(a,m,k-1)$ can be proved for any k. In this context, it is sufficient to verify the criterion

$$\sum_{i=k}^{n} r_{m,i} - \sum_{i=k}^{n} \varphi_{min}(r_{m,i}, x) < \varphi(a,m,k-1) \tag{2.3}$$

for a suitable $x < k$.

This is the fundamental idea of the Discarding Sequential Algorithm (DSA) 4 to compute values of $\omega(n)$. Starting with a sufficiently long empty array and a tentative sequence length m, the parameter m is increased if a longer sequence was found.

It has shown to be more efficient if the first cycles, which employ only small primes, were processed without checking for the possibility of rejection. In later stages of the recursion, say from p_{k^*} on, the criterion can be applied. If the table of φ_{min} includes the corresponding values for $x = k - 1$ then they are preferred. Otherwise, the largest possible x must be used.

Another modification can slightly speed up the algorithm. The criterion 2.3 need not be compared for the entire length m of the considered sequence. It is sufficient to take the maximum length m^* of a subsequence into account which includes all uncovered positions. While counting $\varphi(a, m, k - 1)$, the appropriate m^* can simply be determined as well.

Algorithm 4 Discarding Sequential Algorithm (DSA).

```
procedure DISCARDING_SEQUENTIAL(arr,k)
    for i=1 to plist[k]-1 do
        arr1=arr; fill_array(arr1,i,plist[k])
        if k<n-1 then
            go_on=true
            if k≥ k* then
                if criterion 2.3 fulfilled then go_on=false
                end if
            end if
            if go_on=true then discarding_sequential(arr1,k+1)
            end if
        else count_array(arr1)
            if longer_sequence_found then increase m
            end if
        end if
    end for
end procedure
arr=empty_array                                    ▷ Sequence array
m=starting_sequence_length                         ▷ Starting sequence length
plist=[$p_2$,...,$p_n$]                            ▷ Array of primes
k=1                                                ▷ Starting prime array index
k*=starting_index_for_criterion   ▷ Starting prime array index for criterion
discarding_sequential(arr,k)                       ▷ Recursion
```

Although the DSA algorithm includes an efficient way to recognise inappropriate remainder combinations, it is time-consuming especially for large primes. The number of permutations to be considered in the RPA algorithm, however is much lower than the number of possible remainder combinations. The connection of these two principles in a Combined Reduced Permutation and Discarding Sequential Algorithm (CRPDSA) 5 can make the advantages of both ideas work together [23].

For smaller primes, the DSA algorithm is applied sequentially up to a fixed prime. The remaining primes are handled with the RPA algorithm which was extended by a test of the DSA-criterion. The number of necessary cases can thus rigorously be reduced. An optimum point k^* for the switch between both sub-algorithms was empirically found at $p_n/3$ when p_n is the largest prime considered. We request $p_{k^*}<p_n/3$.

Algorithm 5 Combined Reduced Permutation and Discarding Sequential Algorithm (CRPDSA).

```
procedure COMBINED_DISCARDING(arr,plist,k,qlist)
    if $p_k$<$p_k/3$ then                                                        ▷ DSA part
        for i=1 to plist[k]-1 do
            arr1=arr; fill_array(arr1,i,plist[k]); go_on=true
            if k$\geq$ k$^*$ then
                if criterion 2.3 fulfilled then go_on=false
                end if
            end if
            if go_on=true then combined_discarding(arr1,plist,k+1,qlist)
            end if
        end for
    else
        if empty_qlist then fill_qlist
        end if
        for i=k to n-1 do                                                        ▷ RPA part
            if qlist[k] $\not\equiv$ 0 (mod plist[i]) then
                if forall j<k : qlist[j]$\not\equiv$qlist[k] (mod plist[k]) or plist[j]<plist[k] then
                    arr1=arr; fill_array(arr1,qlist[k],plist[i])
                    if k<n-1 then go_on=true
                        if criterion 2.3 fulfilled then go_on=false
                        end if
                        if go_on=true then
                            plist1=plist; interchange(plist1[k],plist1[i])
                            qlist[k+1]=next_free_position(arr1)
                            combined_discarding(arr1,plist1,k+1,qlist)
                        end if
                    else count_array(arr1)
                        if longer_sequence_found then increase m
                        end if
                    end if
                end if
            end if
        end for
    end if
end procedure
arr=empty_array                                                                  ▷ Sequence array
m=starting_sequence_length                                                       ▷ Starting sequence length
plist=[$p_2$,...,$p_n$]                                                          ▷ Array of primes
qlist=empty_array                                                                ▷ List of first unlabelled positions
k=1                                                                              ▷ Starting prime array index
k$^*$=starting_index_for_criterion                                               ▷ Starting prime array index for criterion
combined_discarding(arr,plist,k,qlist)                                           ▷ Recursion
```

2.4 Counting the actually remaining number of coprimes

The last suggestion of this paper, how to compute $\omega(n)$, is an improvement of the DSA algorithm 4. The actually remaining number of coprimes in the sequence is exactly counted for every pending prime instead of using a general, estimated bound for it. This is more time-consuming for a specific sequence under consideration. On the other hand, exact counts are often lower than the bounds. So, corresponding remainder constellations can be rejected much earlier. This compensates for the time effort spent for counting the exact frequencies.

We redefine some functions of the last subsection in a more general way. They are not restricted to require primes in their ascending order any more. The idea of permutations plays a role again.

Definition 2.3. Let $(\pi_2, \dots, \pi_n)$ be an arbitrary but fixed permutation of $\{p_2, \dots, p_n\}$. For $a \in \mathbb{Z}$, $m \in \mathbb{N}$, $k \in \mathbb{N} \geq 2$, we define

$$\begin{aligned}
&\varphi(a,m,1) = m,\\
&\varphi(a,m,k) = |\{q \in \{1,\dots,m\} \mid \forall\, i \in \{2,\dots,k\} : a+q \not\equiv 0 \ (mod\ \pi_i)\}|, \ \textit{and}\\
&\nu(a,m,k) = \varphi(a,m,k-1) - \varphi(a,m,k).
\end{aligned}$$

The difference $\nu(a,m,k)$ now corresponds to the number of integers coprime to $\pi_2, \dots, \pi_{k-1}$ which are divisible by π_k. Again, let m be the tentative length of the sequence, and $\{a_j\}$ a set of remainders where a is the solution of the simultaneous congruences $a \equiv -a_j \ (mod\ \pi_j)$, $j = 2, \dots, n$. The further processing of the corresponding set of remainders $\{a_j\}$ can then be skipped if

$$\sum_{i=k}^{n} \nu(a,m,i) < \varphi(a,m,k-1)$$

can be proved for any k.

Every position in the sequence corresponds to a definite residue class for each prime. So, the number of remaining coprimes which belong to a residue class can easily be counted. By this means, we derive individual bounds for $\nu(a,m,i)$ depending on the specific choice of $\{a_j\}$.

Definition 2.4. Let $(\pi_2, \dots, \pi_n)$ be an arbitrary but fixed permutation of $\{p_2, \dots, p_n\}$, $m, n, k, t \in \mathbb{N}$ with $2 \leq k < t \leq n$, and $a \in \mathbb{Z}$ with $a \equiv -a_j \ (mod\ \pi_j)$, $j = 2, \dots, k$. Then we define

$$\begin{aligned}
S_1 &= \{1, \dots, m\},\\
S_k &= \{q \in S_{k-1} \mid a+q \not\equiv 0 \ (mod\ \pi_k)\},
\end{aligned}$$

and for $r_t \in \{1, \dots, \pi_t - 1\}$

$$\begin{aligned}
\varrho(a,m,k,t,r_t) &= |\{q \in S_k \mid q \equiv r_t \ (mod\ \pi_k)\}|, \ \textit{and}\\
\varrho_{max}(a,m,k,t) &= \max_{r_t \in \{1,\dots,\pi_k - 1\}} \varrho(a,m,k,t,r_t).
\end{aligned}$$

The set of positions of a sequence $a+1, \dots, a+m$ which are coprime to all primes up to π_k is denoted by S_k, i.e. the set of so far uncovered positions. Given the set S_k, $\varrho(a,m,k,k+1,r_{k+1})$ is the number of positions of the sequence which can be covered in the next step if $a_{k+1} = r_{k+1}$ was chosen. We now relate the function ν to ϱ_{max}.

Lemma 2.4. *Let $a \in \mathbb{Z}$, and $m, n, k, t \in \mathbb{N}$ with $2 \le k < t \le n$. Then*

$$\nu(a, m, t) \le \varrho_{max}(a, m, k, t).$$

Proof. By definition, there exists an $r_t \in \{1, \ldots, \pi_t - 1\}$ so that $\nu(a, m, t) = \varrho(a, m, t-1, t, r_t)$. For $k < t$, we get $\varrho(a, m, t-1, t, r_t) \le \varrho(a, m, k, t, r_t) \le \varrho_{max}(a, m, k, t)$ because $S_k \supseteq S_{t-1}$. □

In consequence of lemma 2.4, it follows

$$\sum_{i=k}^{n} \nu(a, m, i) \le \sum_{i=k}^{n} \varrho_{max}(a, m, k-1, i),$$

and we can now formulate the criterion

$$\sum_{i=k}^{n} \varrho_{max}(a, m, k-1, i) < \varphi(a, m, k-1) \tag{2.4}$$

for the rejection of the further processing of the corresponding set of remainders $\{a_j\}$ for a tentative sequence length m.

The key idea of the effective realisation of this criterion in an algorithm is choosing the residue class with the maximum possible number of newly covered positions $\max_{t \in \{k, \ldots, n\}} \varrho_{max}(a, m, k-1, t)$ in every recursion cycle of level k. This choice implies a permutation algorithm because it selects π_t as the next prime. At the same time, it selects the residue class $r_t \ mod \ \pi_t$ with $\varrho(a, m, k-1, t, r_t) = \max_{t \in \{k, \ldots, n\}} \varrho_{max}(a, m, k-1, t)$. Thus, the so-called Greedy Permutation Algorithm (GPA) 6 combines permutations with respect to primes and to remainders.

After level $k+1$ of recursion was done for that specific choice, the following cycles of level k do not need to consider the consumed $r_t \ mod \ \pi_t$ again because all combinations of it were thereby examined. Otherwise, equivalent permutations would be considered. For the following cycles of level k, exclusively, the corresponding $\varrho(a, m, k-1, t, r_t)$ is set to be 0, and $\varrho_{max}(a, m, k-1, t)$ is determined anew.

Because of the specific choice of the remainders r_t, the Greedy Permutation Algorithm (GPA) is confined to process only prime permutations with $\varrho(a, m, i-1, i, r_i) \le \varrho(a, m, j-1, j, r_j)$ for $i < j$, i.e. the contribution of the primes is monotonically decreasing with the permutation order. Furthermore, a cyclic inner permutation of primes with the same contribution is prevented by blocking those $r_t \ mod \ \pi_t$ always consumed at an earlier level.

It can be shown that the GPA algorithm is again more efficient if the first cycles, which employ only small primes, were processed simply sequentially without any checking for the possibility of rejection. This corresponds to the Basic Sequential Algorithm 1. In later stages of the recursion, say from p_{k^*} on, the new criterion will be applied, and the recursion changes to the true Greedy Permutation Algorithm.

Algorithm 6 Greedy Permutation Algorithm (GPA).

```
procedure GREEDY_PERMUTATION(arr,k,ftab)
    if k<k$^*$ then                                            ▷ Sequential part
        for i=1 to plist[k]-1 do
            arr1=arr; fill_array(arr1,i,plist[k])
            if k=k$^*$-1 then
                ftab1=ftab
                fill_frequency_table_of_remainders(ftab1)
            end if
            greedy_permutation(arr1,k+1,ftab1)
        end for
    else
        if k<n-1 then                                  ▷ Greedy permutation part
            update_$\varphi_{max}(\ldots,i)$
            go_on=true
            if  criterion 2.4 fulfilled then go_on=false
            end if
            if go_on=true then                            ▷ Permutation level k+1
                select_appropriate_$r_t$_and_$\pi_t$
                arr1=arr; fill_array(arr1,$r_t$,$\pi_t$)
                ftab1=ftab
                update_frequency_table_of_remainders(ftab1)
                greedy_permutation(arr1,k+1,ftab1)
            end if
            delete_frequency_of_$r_t$_mod_$\pi_t$(ftab)
            if exists_non-zero_frequency_mod_$\pi_t$(ftab) then   ▷ Permutation level k
                greedy_permutation(arr,k,ftab)
            end if
        else count_array(arr)
            if longer_sequence_found then increase m
            end if
        end if
    end if
end procedure
arr=empty_array                                               ▷ Sequence array
m=starting_sequence_length                           ▷ Starting sequence length
plist=[$p_2$,...,$p_n$]                                         ▷ Array of primes
k=1                                                 ▷ Starting prime array index
k$^*$=starting_index_for_criterion     ▷ Starting prime array index for criterion
ftab=empty_table                               ▷ Frequency table of remainders
greedy_permutation(arr,k,ftab)                                      ▷ Recursion
```

2.5 Parallel processing

All depicted explicit algorithms can be processed in parallel in a simple way. All of them are organised as treelike recursive procedures. Given a fixed recursion level k^*, the recurrent computation can be prepared in a first step by executing the recursive procedures until level k^*, only. All relevant parameters defining the next procedure call for level $k^* + 1$ are written to a separate parameter file. Now in a second step, the subsequent subtrees or series of them, each starting at level $k^* + 1$, can be processed to their end on different cores, processors, or even on different computers. The last step summarises all individual results of these subprocesses, including all sequences of maximum length.

Parallel computation can be used most efficiently by a prior sorting of the subtrees of level $k^* + 1$ by descending $\varphi(a, m, k)$. This leads to an earlier recognition of longer sequences on average. The rapid enlargement of the respective array prevents from examining many too small sequences unnecessarily, and therefore speeds up the entire approach.

3 Results

The algorithms described in section 2 act very differently concerning its computational effort. We computed the entire values of the function $h(n)$ where all involved primes could be represented as single byte, i.e. $p_n \leq 251$. In these calculations, we used several algorithms each suitable regarding computation time. The equality of corresponding results served as an implicit verification of the correctness of the implementations. Our data are also in accord with all published data for $p_n \leq 227$ [5].

For every investigated n, we performed an exhaustive retrieval for all existing sequences of maximum length while searching for $\omega(n)$. As far as we know, this has not been done ever before. We provide this data in ancillary files.

3.1 Calculated data

All algorithms output values of the function $\omega(n)$ according to definition 1.5 as their main result. In addition, every compatible sequence of the appropriate length $\omega(n)$ was recorded. The values of $h(n)$, see definition 1.3, were deduced from $\omega(n)$ by applying corollary 1.2.

An initial illustration of the results is outlined in figure 1. We graph the values for h(n) and number of sequences of length $\omega(n)$. This number considerably varies in a non-obvious manner.

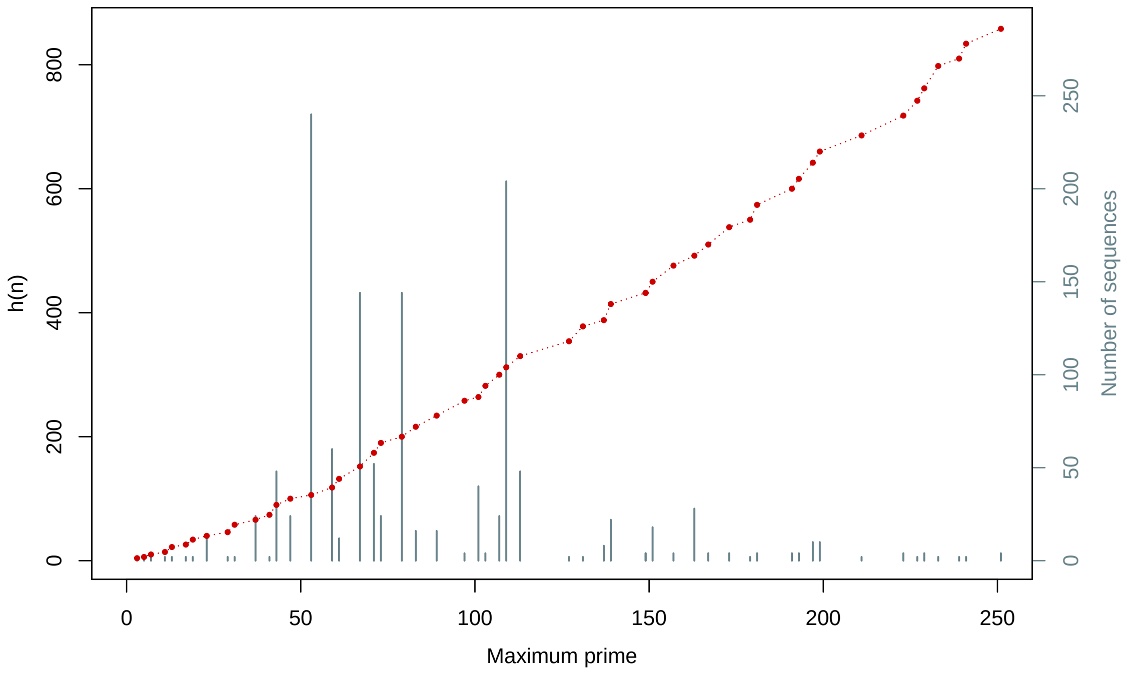

Figure 1: Values of the function $h(n)$ and the number of respective sequences of maximum length.

The complete data are provided in table 1 including the maximum processed prime p_n, the values of $h(n)$ and $\omega(n)$, and the number of sequences n_{seq} for every $n \leq 54$.

n	p_n	h(n)	$\omega(n)$	n_{seq}
1	2	–	–	–
2	3	4	1	1
3	5	6	2	2
4	7	10	4	2
5	11	14	6	2
6	13	22	10	2
7	17	26	12	2
8	19	34	16	2
9	23	40	19	12
10	29	46	22	2
11	31	58	28	2
12	37	66	32	24
13	41	74	36	2
14	43	90	44	48
15	47	100	49	24
16	53	106	52	240
17	59	118	58	60
18	61	132	65	12
19	67	152	75	144
20	71	174	86	52
21	73	190	94	24
22	79	200	99	144
23	83	216	107	16
24	89	234	116	16
25	97	258	128	4
26	101	264	131	40
27	103	282	140	4
28	107	300	149	24
29	109	312	155	204
30	113	330	164	48
31	127	354	176	2
32	131	378	188	2
33	137	388	193	8
34	139	414	206	22
35	149	432	215	4
36	151	450	224	18
37	157	476	237	4
38	163	492	245	28
39	167	510	254	4
40	173	538	268	4
41	179	550	274	2
42	181	574	286	4
43	191	600	299	4
44	193	616	307	4
45	197	642	320	10
46	199	660	329	10
47	211	686	342	2
48	223	718	358	4
49	227	742	370	2
50	229	762	380	4
51	233	798	398	2
52	239	810	404	2
53	241	834	416	2
54	251	858	428	4

Table 1: Computation results.

3.2 Computation time

All algorithms except ILP were implemented and executed on a common PC with an i7 processor at boosted 3.8GHz in a single thread application. For solving the respective integer linear programs according to the equations 2.2, however, we utilised SYMPHONY software [18, 14] which used all threads in parallel. The corresponding time needs were enlarged by an empirically estimated factor to make them comparable.

The time consumption of each algorithm rapidly grows with the number of primes. Figure 2 depicts this growth as well as the variation between different algorithms. A quasi-logarithmic time scale $log(1+t)$ was applied where t was the exact time need rounded to full seconds.

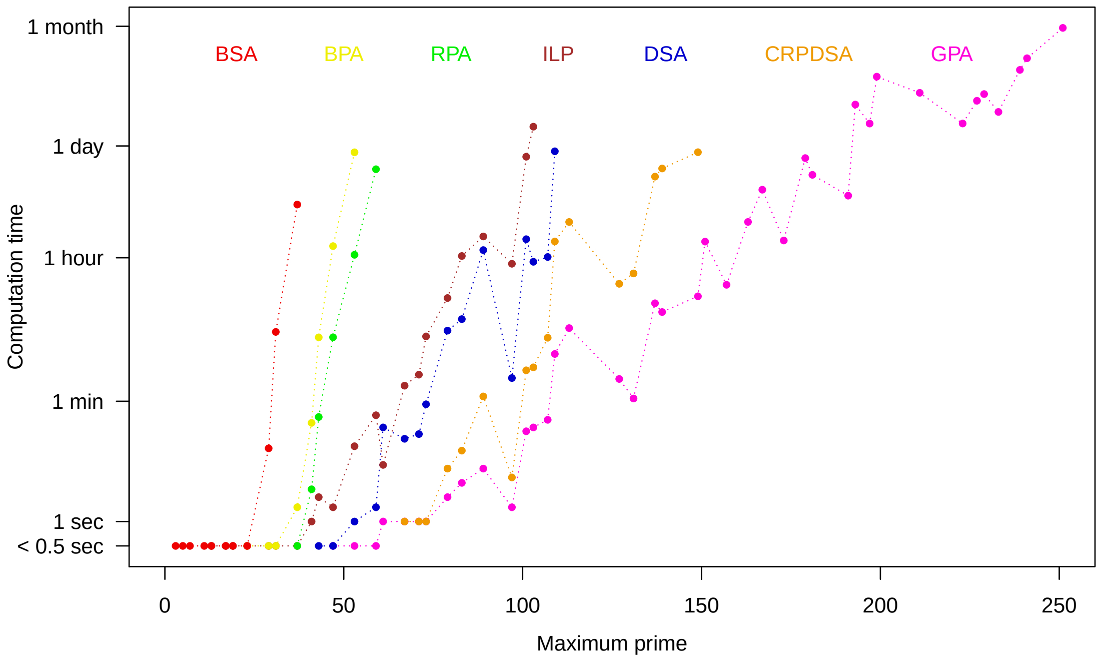

Figure 2: Comparison of the computational cost for various algorithms.

3.3 Ancillary data

In addition to this paper, we provide four files including the complete results of our calculations. The first file presents intrinsic data of the DSA algorithm 4.

phi_min.txt

This file contains a table of values of the function $\phi_{min}(m,k)$ as described in definition 2.2 for $m \leq 500$ and $k \leq 11$. The data were computed using a brute force approach very similar to the Basic Sequential Algorithm 1. While BSA intends to maximise the number of covered positions in a given array, the computation of $\varphi_{min}(m,k)$ needs to minimise it.

The other ancillary files contain exhaustive lists of all sequences of the appropriate maximum lengths. According to propositions 1.3 and 1.5, these sequences can be represented in three ways as always has been concluded in remark 4.

moduli.txt

This file contains the modulus-representation of the sequences. In remark 4, paragraph (1), every position $q \in \{1, \ldots, m\}$ of the sequence was related to at least one prime modulus $p_i,\ i \in \{2, \ldots, n\}$. The progression of primes $p_q,\ q \in \{1, \ldots, m\}$ of the minimum appropriate moduli p_q each position q is a unique representation of the sequence under consideration.

remainders.txt

This file contains the remainder-representation of the sequences, i.e. the ordered set of remainders $a_i\ (mod\ p_i),\ i \in \{2, \ldots, n\}$ as described in remark 4, paragraph (2).

permutations.txt

This file contains the permutation-representation of the sequences, i.e. the permutation of primes $\pi_i,\ i \in \{2, \ldots, n\}$ as described in remark 4, paragraph (3).

The sequences in "remainders.txt" are separately sorted for each n by ascending remainders. This order was maintained in the other files "moduli.txt" and "permutations.txt" to make a direct comparison possible..

3.4 Final remarks

All depicted algorithms may also be applied to arbitrary sets of different primes. The specific choice of consecutive primes was not necessarily required. With the help of prime separation as described in remark 1, all values of the original Jacobsthal function $j(n)$ can therefore be computed using a generalised implementation of one of these algorithms.

Acknowledgement

The authors would like to express their appreciation to the On-Line Encyclopedia of Integer Sequences [17]. It provides an exceptional collection of integer progressions. This data is very informative and helps discover new items and links within the wide field of number theory.

Furthermore, the work of the COIN-OR project [14] is highly acknowledged. Modern approaches in computational operations research are developed, implemented, and disseminated in open source software.

Contact

marioziller@arcor.de
axelmorack@live.com

References

[1] Fintan Costello and Paul Watts, *A computational upper bound on Jacobsthal's function*, arXiv:1208.5342 [math.NT] (2012).

[2] Fintan Costello and Paul Watts, *A short note on Jacobsthal's function*, arXiv:1306.1064 [math.NT] (2013).

[3] Paul Erdös, *On the Integers Relatively Prime to n and a Number-Theoretic Function Considered by Jacobsthal*, MATHEMATICA SCANDINAVICA **10** (1962), 163–170.

[4] Daniel M. Gordon and Eugene R. Rodemich, *Dense admissible sets*, Algorithmic Number Theory, Third International Symposium, ANTS-III, Portland, Oregon, USA, June 21-25, 1998, Proceedings (1998), pp. 216–225.

[5] Thomas R. Hagedorn, *Computation of Jacobsthal's Function h(n) for n < 50*, Mathematics of Computation **78** (2009), no. 266, 1073–1087.

[6] Jan Kristian Haugland, *The On-Line Encyclopedia of Integer Sequences [17]*, Sequence A048669 (1999), http://oeis.org/A048669.

[7] Jan Kristian Haugland, *The On-Line Encyclopedia of Integer Sequences [17]*, Sequence A048670 (1999), http://oeis.org/A048670.

[8] Ernst Jacobsthal, *Über Sequenzen ganzer Zahlen, von denen keine zu n teilerfremd ist. I*, D.K.N.V.S. Forhandlinger **33** (1960), no. 24, 117–124.

[9] Ernst Jacobsthal, *Über Sequenzen ganzer Zahlen, von denen keine zu n teilerfremd ist. II*, D.K.N.V.S. Forhandlinger **33** (1960), no. 25, 125–131.

[10] Ernst Jacobsthal, *Über Sequenzen ganzer Zahlen, von denen keine zu n teilerfremd ist. III*, D.K.N.V.S. Forhandlinger **33** (1960), no. 26, 132–139.

[11] Ernst Jacobsthal, *Über Sequenzen ganzer Zahlen, von denen keine zu n teilerfremd ist. IV*, D.K.N.V.S. Forhandlinger **34** (1961), no. 1, 1–7.

[12] Ernst Jacobsthal, *Über Sequenzen ganzer Zahlen, von denen keine zu n teilerfremd ist. V*, D.K.N.V.S. Forhandlinger **34** (1961), no. 24, 110–115.

[13] Michael Kleber, *The On-Line Encyclopedia of Integer Sequences [17]*, Sequence A132468 (2007), http://oeis.org/A132468.

[14] Robin Lougee-Heimer, *The Common Optimization INterface for Operations Research: Promoting Open-source Software in the Operations Research Community*, IBM J. Res. Dev. **47** (2003), no. 1, 57–66.

[15] Jud McCranie, *The On-Line Encyclopedia of Integer Sequences [17]*, Sequence A058989 (2001), http://oeis.org/A058989.

[16] John F. Morack, *Contribution to OEIS A072752 [20]*, private communication (2014), http://oeis.org/A072752.

[17] OEIS Foundation Inc., *The On-Line Encyclopedia of Integer Sequences*, (2011), http://oeis.org.

[18] Ted Ralphs, Menal Guzelsoy, Ashutosh Mahajan, and Laszlo Ladanyi, *SYMPHONY*, version 5.6.14 (2016), https://projects.coin–or.org/SYMPHONY.

[19] Giovanni Resta, *Contribution to OEIS A072753 [21]*, private communication (2015), http://oeis.org/A072753.

[20] Mario Ziller, *The On-Line Encyclopedia of Integer Sequences [17]*, Sequence A072752 (2002), http://oeis.org/A072752.

[21] Mario Ziller, *The On-Line Encyclopedia of Integer Sequences [17]*, Sequence A072753 (2002), http://oeis.org/A072753.

[22] Mario Ziller, *Contribution to OEIS A072752 [20]*, private communication (2005), http://oeis.org/A072752.

[23] Mario Ziller and John F. Morack, *Contribution to OEIS A072752 [20]*, private communication (2005), http://oeis.org/A072752.