Computation of numerical semigroups by means of seeds

Maria Bras-Amorós, Julio Fernández-González

April 23, 2018

Abstract

For the elements of a numerical semigroup which are larger than the Frobenius number, we introduce the definition of *seed* by broadening the notion of generator. This new concept allows us to explore the semigroup tree in an alternative efficient way, since the seeds of each descendant can be easily obtained from the seeds of its parent. The paper is devoted to presenting the results which are related to this approach, leading to a new algorithm for computing and counting the semigroups of a given genus.

Introduction

Let $\mathbb{N}_0$ be the set of non-negative integers. A *numerical semigroup* is a subset Λ of $\mathbb{N}_0$ which contains 0, is closed under addition and has finite complement, $\mathbb{N}_0 \backslash \Lambda$. The elements in $\mathbb{N}_0 \backslash \Lambda$ are the *gaps* of Λ, and the number $g = g(\Lambda)$ of gaps is the *genus* of Λ. Thus, the unique increasing bijective map

$$\begin{array}{ccc} \mathbb{N}_0 & \longrightarrow & \Lambda \\ i & \longmapsto & \lambda_i \end{array}$$

indexing the *non-gaps*, namely the elements in Λ, satisfies $\lambda_0 = 0$ and $\lambda_i = i + g$ for any large enough positive index i. Let $k = k(\Lambda)$ denote the smallest such index. The *conductor* of Λ is

$$c = c(\Lambda) = \lambda_k = k + g.$$

The trivial semigroup $\mathbb{N}_0$ corresponds to $c = 1$, that is, to $g = 0$. Otherwise, the largest gap of Λ, which is known as its *Frobenius number*, is $c - 1$. We call Λ *ordinary* whenever c equals the *multiplicity* λ_1, that is, whenever $g = c - 1$.

A *generator* of Λ is a non-gap σ such that $\Lambda \backslash \{\sigma\}$ is still a semigroup, which amounts to saying that σ is not the sum of two non-zero non-gaps. Any semigroup of genus $g \geq 1$ can be uniquely obtained from a semigroup Λ of genus $g - 1$ by removing a generator $\sigma \geq c(\Lambda)$. This allows one to arrange all numerical semigroups in an infinite tree, rooted at the trivial semigroup, such that the nodes at depth g are all semigroups of genus g. This tree was already introduced in [19, 20] and later considered in [6, 7, 9]. The first nodes are shown in Figure 1, where each semigroup is represented by its non-zero elements up to the conductor.

The first author was supported by the Spanish government under grant TIN2016-80250-R and by the Catalan government under grant 2014 SGR 537.
The second author is partially supported by the Spanish government under grant MTM2015-66180-R.

The number n_g of semigroups of genus g was conjectured in [6] to satisfy

$$n_{g+2} \geq n_{g+1} + n_g$$

for $g \geq 0$, and to behave asymptotically like the Fibonacci sequence. Zhai proved the second statement in [22]. The *Sloane's On-line Encyclopedia of Integer Sequences* [21] stores the known portion of the sequence n_g. The first values have been computed by Medeiros and Kakutani, Bras-Amorós [6], Delgado [10], and Fromentin and Hivert [13]. Several contributions [7, 9, 11, 23, 2, 3, 15, 8, 17] have been made to theoretically analyze the sequence.

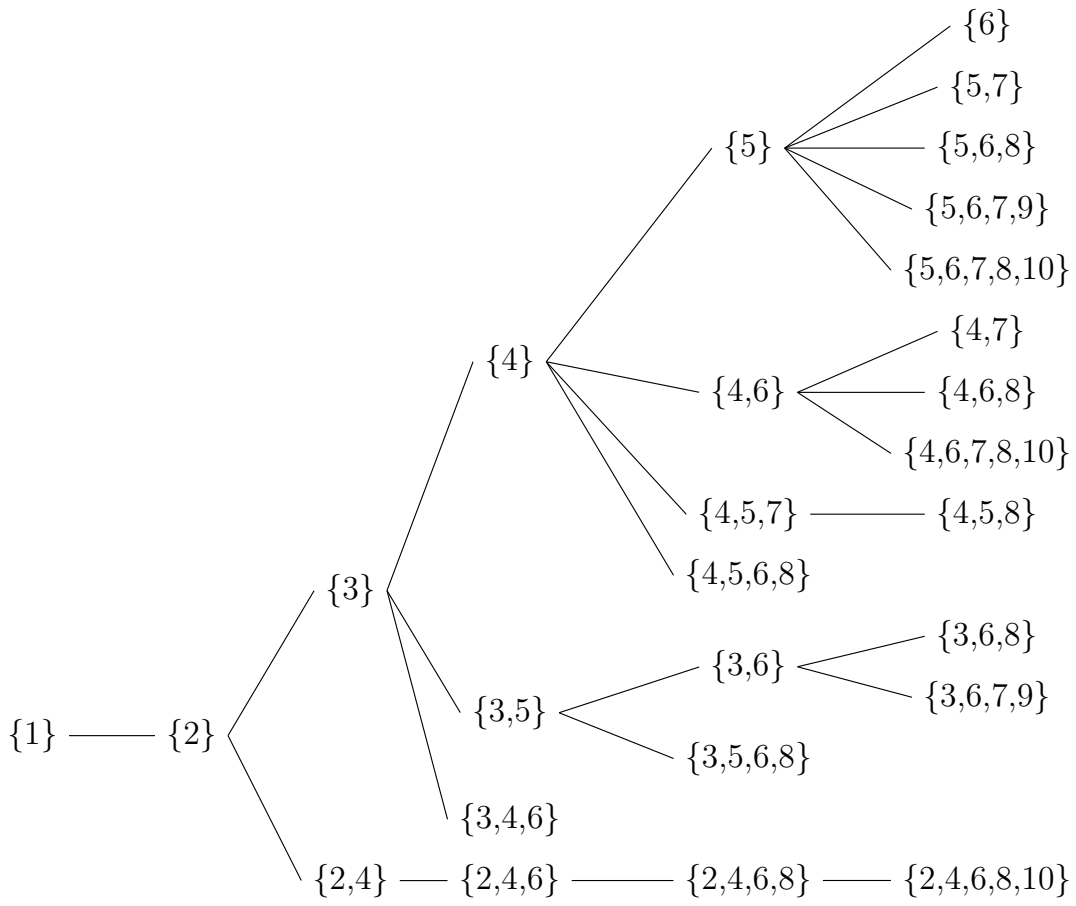


Figure 1

In this article we introduce the notion of *seeds*, as a generalization of the generators of a semigroup which are greater than the Frobenius number. This new concept allows us to explore the semigroup tree in an alternative efficient way, since the seeds of each descendant can be easily obtained from the seeds of its parent. Sections 1 and 2 are devoted to presenting the results which are related to this approach, leading in Section 3 to a new algorithm for computing the (number of) semigroups of a given genus. The running times that we obtain are shorter than for the other algorithms which can be found in the literature.

1 Seeds of a numerical semigroup

Let us fix a numerical semigroup $\Lambda = \{\lambda_i\}_{i\geq 0}$ with conductor $c = \lambda_k$ and genus g. For any index $i \geq 0$, the set

$$\Lambda_i := \Lambda \setminus \{\lambda_1, \ldots, \lambda_i\}$$

is a semigroup of genus $g+i$. It has conductor c if and only if $i \leq k-1$, and it is ordinary if and only if $i \geq k-1$.

Definition 1.1. An element λ_t with $t \geq k$ is a *seed* of Λ if $\lambda_t + \lambda_i$ is a generator of Λ_i for some index $0 \leq i < k$. We then say that the seed λ_t has *order* i, and also that λ_t is an *order-i seed*.

A seed of Λ is, by definition, at least as large as the conductor, and it has order zero precisely when it is a generator of Λ. The order-zero seeds of a semigroup of genus g are in bijection with its descendants of genus $g+1$ in the semigroup tree, which are obtained by removing exactly one of those seeds. Order-one seeds have been named *strong generators* in other references, such as [9]. They are behind the key idea of the bounds in [7].

Lemma 1.2. *Any order-i seed of Λ is at most $c + \lambda_{i+1} - \lambda_i - 1$. In particular, the number of order-i seeds of Λ is at most $\lambda_{i+1} - \lambda_i$.*

Proof. A generator of the semigroup Λ_i must be smaller than the sum of the conductor, which is c for $0 \leq i < k$, and the multiplicity, which is λ_{i+1}. □

Definition 1.3. The *table of seeds* of Λ is a binary table whose rows are indexed by the possible seed orders. For $0 \leq i \leq k-1$, the i-th row has $\lambda_{i+1} - \lambda_i$ entries, each of them corresponding to a possible order-i seed of Λ. They are defined as follows: for $0 \leq j \leq \lambda_{i+1} - \lambda_i - 1$, the j-th entry in the i-th row is 1 if $\lambda_{k+j} = c + j$ is an order-i seed of Λ, and 0 otherwise. The total number of entries in the table is c.

Example 1.4. For the semigroup

$$\Lambda \;=\; \mathbb{N}_0 \setminus \{1,2,3,4,6,7\} \;=\; \{0,5,8,9,10,\ldots\}$$

one has $\lambda_1 = 5$, $c = \lambda_2 = 8$, so Λ may have only order-zero and order-one seeds. The former are simply the generators of Λ among the elements

$$\lambda_2 = 8, \quad \lambda_3 = 9, \quad \lambda_4 = 10, \quad \lambda_5 = 11, \quad \lambda_6 = 12,$$

whereas the latter may only be among the first three, according to Lemma 1.2. Although $\lambda_4 = 2\lambda_1$ is clearly the only one of these five elements which is not a generator of Λ, it is an order-one seed because $\lambda_4 + \lambda_1 = 15$ is not the sum of two non-zero elements in the semigroup

$$\Lambda_1 \;=\; \Lambda \setminus \{\lambda_1\} \;=\; \mathbb{N}_0 \setminus \{1,2,3,4,5,6,7\} \;=\; \{0,8,9,10,\ldots\}.$$

Since Λ_1 is ordinary in this example, $\lambda_2 + \lambda_1$ and $\lambda_3 + \lambda_1$ must then also be generators of this semigroup, which means that both λ_2 and λ_3 are also order-one seeds. Thus, the table of seeds of Λ is as follows:

1	1	0	1	1
1	1	1		

Example 1.5. For the semigroup

$$\Lambda = \mathbb{N}_0 \setminus \{1,2,3,4,5,6,7,9,12,13\} = \{0,8,10,11,14,15,16,\dots\}$$

one has $\lambda_1 = 8$, $\lambda_2 = 10$, $\lambda_3 = 11$, $c = \lambda_4 = 14$. According to Lemma 1.2, Λ may have at most 8, 2, 1 and 3 order-i seeds for $i = 0,1,2,3$, respectively, and the possible seeds for each order are given consecutively from λ_4 on. The actual seeds may be sieved from scratch using Definition 1.1, as in the previous example. Specifically, the table of seeds of Λ can be checked to be as follows:

1	1	0	1	0	0	0	0
0	1						
1							
1	1	1					

For instance, λ_4 is not an order-one seed because $\lambda_4 + \lambda_1 = 22 = 2\lambda_3$ is not a generator of the semigroup

$$\Lambda_1 = \Lambda \setminus \{\lambda_1\} = \{0,10,11,14,15,16,\dots\},$$

whereas it is an order-two seed because $\lambda_4 + \lambda_2 = 24$ is not the sum of two non-zero elements in the semigroup

$$\Lambda_2 = \Lambda \setminus \{\lambda_1, \lambda_2\} = \{0,11,14,15,16,\dots\}.$$

2 Behavior of seeds along the semigroup tree

Our aim in this section is to deduce the table of seeds of a descendant in the semigroup tree from the table of seeds of its parent. We keep the notations from the previous section.

Let us fix an order-zero seed λ_s of Λ, so that $s \geq k$ and

$$\tilde{\Lambda} := \Lambda \setminus \{\lambda_s\}$$

is a semigroup of genus $g+1$. The elements in $\tilde{\Lambda}$ are

$$\tilde{\lambda}_i = \lambda_i \quad \text{for} \quad 0 \leq i < s, \qquad \tilde{\lambda}_i = \lambda_{i+1} = \lambda_i + 1 \quad \text{for} \quad i \geq s,$$

and the conductor is

$$c(\tilde{\Lambda}) = \tilde{\lambda}_s = \lambda_s + 1 = s + g + 1.$$

So there are s possible seed orders for this semigroup: an order-i seed of $\tilde{\Lambda}$, with $0 \leq i < s$, is an element λ_t with $t > s$ such that $\lambda_t + \lambda_i$ is a generator of the semigroup

$$\tilde{\Lambda}_i := \tilde{\Lambda} \setminus \{\tilde{\lambda}_1, \dots, \tilde{\lambda}_i\} = \Lambda \setminus \{\lambda_1, \dots, \lambda_i, \lambda_s\}.$$

Clearly, any order-i seed λ_t of Λ with $t > s$ is also an order-i seed of $\tilde{\Lambda}$. This corresponds to the first type of *old-order seeds* of $\tilde{\Lambda}$, meaning by this term the order-i seeds of $\tilde{\Lambda}$ with $0 \leq i < k$. The four types of such seeds are gathered in the next result.

Lemma 2.1. *Let i be an index with $0 \leq i < k$. An element λ_t with $t > s$ is an order-i seed of $\tilde{\Lambda}$ if and only if one of the following pairwise excluding conditions holds:*

(1) *λ_t is an order-i seed of Λ.*

(2) *$i < k-1$, $\lambda_t = \lambda_s + \lambda_{i+1} - \lambda_i$ and λ_s is an order-$(i+1)$ seed of Λ.*

(3) *$i = k-1 = s-2$ and $\lambda_t = \lambda_s + \lambda_k - \lambda_{k-1}$.*

(4) *$i = k-1 = s-1$ and either $\lambda_t = \lambda_s + \lambda_k - \lambda_{k-1}$ or $\lambda_t = \lambda_s + \lambda_k - \lambda_{k-1} + 1$.*

Proof. Let us first assume that λ_t is an order-i seed of $\tilde{\Lambda}$. Then, Lemma 1.2 and the definition of $\tilde{\Lambda}$ yield the inequality

$$\lambda_t \;\leq\; \lambda_s \;+\; \tilde{\lambda}_{i+1} \;-\; \lambda_i.$$

Furthermore, (1) is not satisfied if and only if

$$\lambda_t \;=\; \lambda_s \;+\; \lambda_j \;-\; \lambda_i$$

for some index $j \geq i+1$. If this holds, then $\lambda_{i+1} \leq \lambda_j \leq \tilde{\lambda}_{i+1}$, where $\tilde{\lambda}_{i+1} = \lambda_{i+1}$ if $i < s-1$, while $\tilde{\lambda}_{i+1} = \lambda_{i+1} + 1$ otherwise. In the first case, one has $j = i+1$, which leads to the expression for λ_t displayed in (2) and (3). Since $0 \leq i < k \leq s$, the second case can only occur for $i = s-1 = k-1$, and then the only two possibilities for λ_t are those in (4).

Let us now examine when the element $\lambda_s + \lambda_{i+1} - \lambda_i$ is an order-i seed of $\tilde{\Lambda}$. It suffices to remark that the sum $\lambda_s + \lambda_{i+1}$ is a generator of $\tilde{\Lambda}_i$ if and only if it is a generator of Λ_{i+1}. Then, two cases must be distinguished, depending on whether $i < k-1$ or $i = k-1$. In the first case, the latter condition on λ_s amounts to the last one in (2) by definition of seed. As for the second case, $\lambda_s + \lambda_k$ is a generator of Λ_k if and only if s and k are related as in (3) or (4), since otherwise it can be written as the sum of two elements in that semigroup, namely λ_{s-1} and λ_{k+1}.

To conclude the proof, one only needs to check the second possibility in (4). Indeed, $2\lambda_s + 1$ is a generator of the ordinary semigroup $\tilde{\Lambda}_{s-1} = \Lambda_s$. □

Assume now that the removed seed λ_s is different from the conductor of Λ, that is, $s > k$. If $s = k+1$, then there are exactly two *new-order seeds* of $\tilde{\Lambda}$, meaning by this term the order-i seeds of $\tilde{\Lambda}$ with $k \leq i < s$. Otherwise, there are always three such seeds. This follows right away from the next result.

Lemma 2.2. *For an index i at least k, the following hold:*

- *If $i < s-2$, then $\tilde{\Lambda}$ has no order-i seeds.*
- *If $i = s-2$, then the only order-i seed of $\tilde{\Lambda}$ is $\lambda_s + 1$.*
- *If $i = s-1$, then the only order-i seeds of $\tilde{\Lambda}$ are $\lambda_s + 1$ and $\lambda_s + 2$.*

Proof. Let λ_t be an order-i seed of $\tilde{\Lambda}$, so that $i < s < t$. Since $i \geq k$, one has

$$\lambda_i \;=\; \lambda_{i+1} \;-\; 1 \;=\; \lambda_{i+2} \;-\; 2 \qquad \text{and} \qquad \lambda_t \;=\; \lambda_{t-1} \;+\; 1 \;=\; \lambda_{t-2} \;+\; 2.$$

Then both $s-i$ and $t-s$ are at most 2. Otherwise, $\lambda_t + \lambda_i$ could be written as the sum of two elements in the semigroup $\tilde{\Lambda}_i$ above: either $\lambda_{t-1} + \lambda_{i+1}$ or $\lambda_{t-2} + \lambda_{i+2}$. Furthermore, $t = s+1$ must hold whenever $s = i+2$.

For $i = s-2$ and $i = s-1$, one has, respectively,

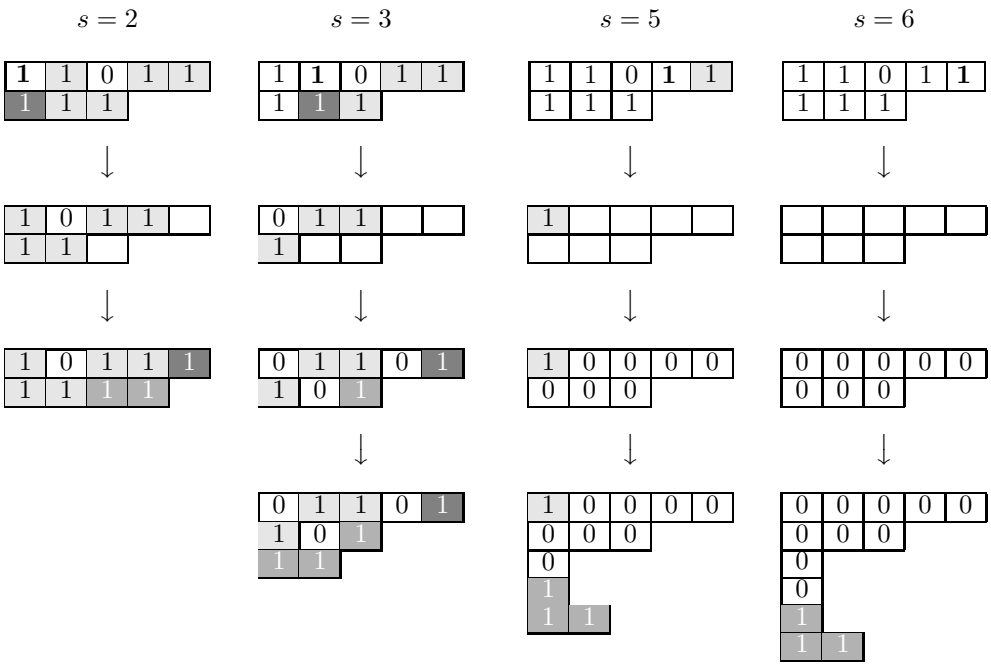
$$\tilde{\Lambda}_i = \mathbb{N}_0 \setminus \{1, \ldots, \lambda_s - 2, \lambda_s\} \qquad \text{and} \qquad \tilde{\Lambda}_i = \mathbb{N}_0 \setminus \{1, \ldots, \lambda_s\}.$$

The proof concludes by checking that $\lambda_{s+1} + \lambda_i = 2\lambda_s - 1$ is a generator of $\tilde{\Lambda}_i$ in the first case and that $\lambda_{s+1} + \lambda_i = 2\lambda_s$ and $\lambda_{s+2} + \lambda_i = 2\lambda_s + 1$ are both generators of $\tilde{\Lambda}_i$ in the second case. □

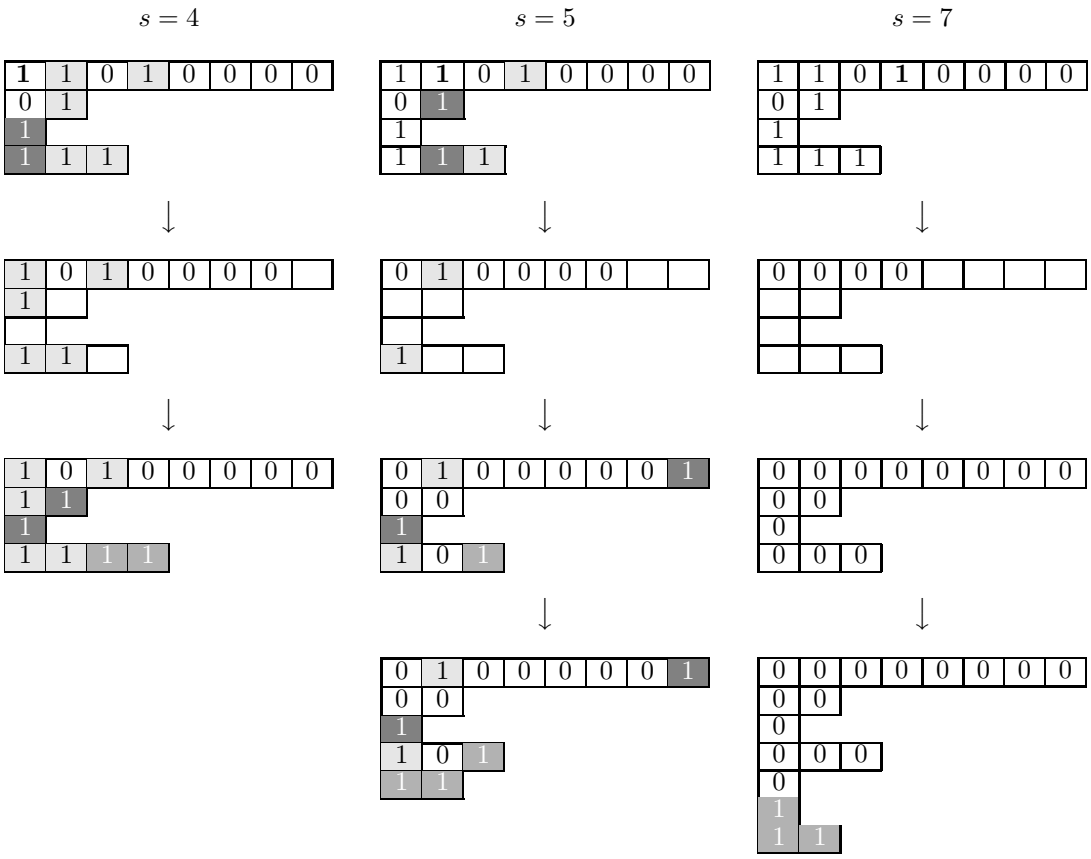
Now the table of seeds of $\tilde{\Lambda}$ can be easily built from the table of seeds of Λ. According to the results in this section, this may be done as follows:

- In each of the k rows of the table, $s-k+1$ new entries must be added to the right, and the same number of entries must be cropped starting from the left. This corresponds to recycling old-order seeds of type (1).
- Then, the last right-hand new entry in the last row must be set to 1 exactly when $s=k$ or $s=k+1$. Moreover, in the first case an additional right-hand entry with value 1 is required. The last right-hand new entry in any other row must be given the value, if any, of the last right-hand cropped entry in the row just below. All entries which remain empty are then set to 0. This codifies old-order seeds of types (2), (3) and (4), and completes all modifications to be done in the first k rows.
- Finally, to take account of new-order seeds, $s-k$ rows must be added to the table if $s > k$. The last of them consists of two entries with value 1. If $s > k+1$, the last but one row consists of a single entry with value 1, while each of the remaining new rows consists of a single entry with value 0.

Example 2.3. We can compute the tables of seeds corresponding to the four descendants of the semigroup Λ in Example 1.4, that is, the table of seeds of $\tilde{\Lambda}$ for $s = 2, 3, 5, 6$, by graphically implementing in each case the three steps above:



Example 2.4. We can compute the table of seeds for each of the descendants of the semigroup Λ in Example 1.5, that is, the table of seeds of $\tilde{\Lambda}$ for $s = 4, 5, 7$.



3 The seeds descending algorithm

Let us associate with a numerical semigroup Λ two binary strings

$$G(\Lambda) \,=\, G_0\, G_1\, \cdots\, G_\ell\, \cdots \qquad \text{and} \qquad S(\Lambda) \,=\, S_0\, S_1\, \cdots\, S_\ell\, \cdots$$

encoding, respectively, the gaps and the seeds of Λ. Specifically, with the notations and definitions in Section 1,

$$G_\ell := \begin{cases} 1 & \text{if } \ell+1 \text{ is a gap of } \Lambda, \\ 0 & \text{otherwise,} \end{cases}$$

whereas $S(\Lambda)$ amounts to the concatenation of the rows in the table of seeds of Λ, that is,

$$S_{\lambda_i+j} := \begin{cases} 1 & \text{if } c+j \text{ is an order-}i \text{ seed of } \Lambda, \\ 0 & \text{otherwise,} \end{cases}$$

for $i \geq 0$ and $0 \leq j < \lambda_{i+1} - \lambda_i$. In particular, $G_\ell = S_\ell = 0$ for $\ell \geq c$, so below we identify each string with its first c bits:

$$G(\Lambda) \,=\, G_0\, G_1\, \cdots\, G_{c-1} \qquad \text{and} \qquad S(\Lambda) \,=\, S_0\, S_1\, \cdots\, S_{c-1}.$$

The bit G_{c-1} is always 0, while $S_{c-3} = S_{c-2} = S_{c-1} = 1$ whenever $c \geq 3$. The pairs of binary strings $G(\Lambda)$, $S(\Lambda)$, for Λ running over the first nodes in the semigroup tree, are displayed in Figure 2.

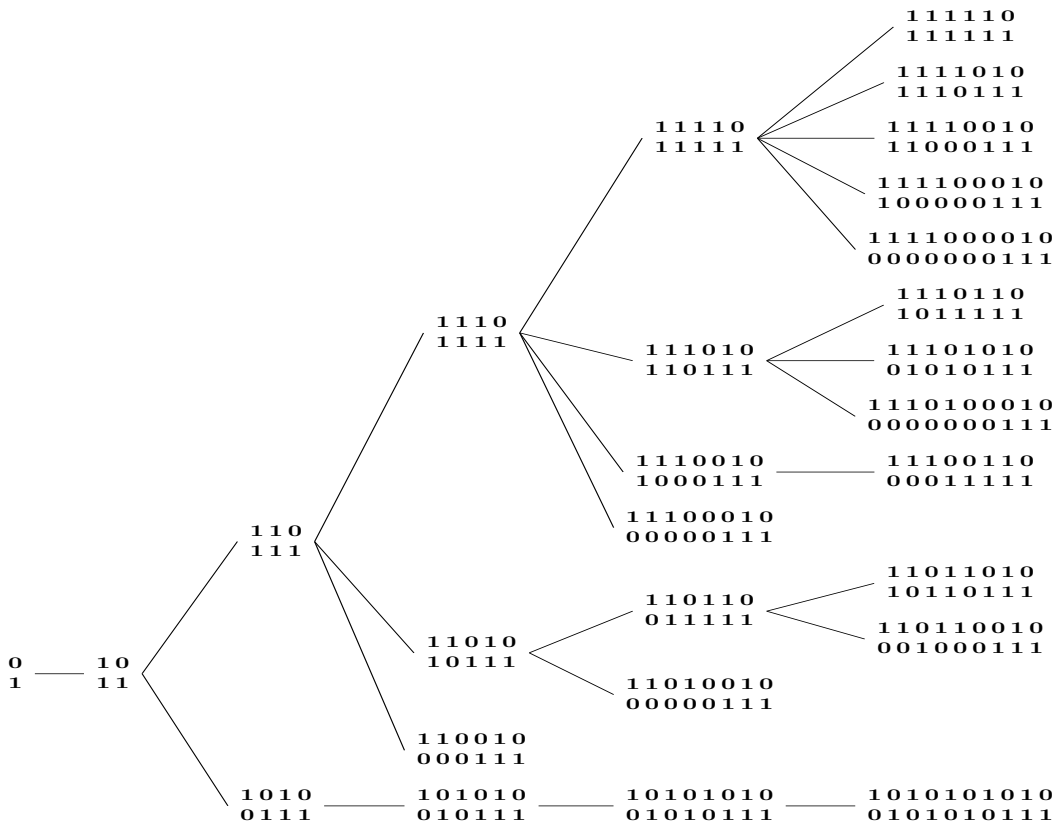


Figure 2

When transcribing and implementing the pseudocodes in this section, binary strings such as $G(\Lambda)$ and $S(\Lambda)$ are regarded, stored and manipulated as integers which are encoded in binary form. We use below the bitwise operations on binary strings & (*and*) and | (inclusive *or*), as well as the *left shift* $\ll$ and the *right shift* $\gg$ of a binary string by a non-negative integer x. In terms of binary encoding of integers, the latter shift is equivalent to multiplying by 2^x.

Assume from now on that Λ is not the trivial semigroup and let $\tilde{\Lambda} = \Lambda \backslash \{\lambda_s\}$ be a descendant of Λ in the semigroup tree, as in Section 2. Then set $\tilde{s} = s - k$, so that $0 \le \tilde{s} < \lambda_1$ and $\lambda_s = c + \tilde{s}$. The string $G(\tilde{\Lambda})$ is obtained from $G(\Lambda)$ by simply switching the bit $G_{c+\tilde{s}-1}$ from 0 to 1. As for $S(\tilde{\Lambda})$, it can be computed from $S(\Lambda)$ by adapting the construction of the table of seeds of $\tilde{\Lambda}$ as is described in Section 2, which leads to the following rephrasing in terms of strings.

Lemma 3.1. *Let* $\tilde{S} = \tilde{S}_0\, \tilde{S}_1 \cdots \tilde{S}_\ell \cdots$ *be the binary string defined by*

$$\tilde{S}_\ell := \begin{cases} 0 \quad \text{if} \quad \ell = \lambda_i + j \quad \text{with} \quad 1 \le i < k,\ 0 \le j < \min(\tilde{s},\, \lambda_{i+1} - \lambda_i), \\ S_\ell \quad \text{otherwise.} \end{cases}$$

Then,

$$S(\tilde{\Lambda}) \;=\; (\tilde{S} \ll \tilde{s} + 1) \,|\, (1\,1\,1 \gg c + \tilde{s} - 2) \;=\; \tilde{S}_{\tilde{s}+1}\, \tilde{S}_{\tilde{s}+2} \cdots \tilde{S}_{c-1}\, \overbrace{0 \cdots 0\; 1}^{2\tilde{s}}\, 1\,1.$$

In particular, $S(\tilde{\Lambda}) = S_1 \cdots S_{c-1}\, 1\,1$ *whenever* $\tilde{s} = 0$.

The string $\tilde{S}$ in this result satisfies $\tilde{S}_{\lambda_i + j} = 0$ for $1 \le i < k,\ 0 \le j < \tilde{s},$ so it can be obtained by *raking* $S(\Lambda)$ to get rid of the old-order seeds which must not

be recycled. This *raking* process is performed by means of successive one-position shifts of $G(\Lambda)$ to the right, as shown in the following pseudocode:

Input: $c := c(\Lambda),\ \ G := G(\Lambda),\ \ S := S(\Lambda),\ \ \tilde{s}$

Output: $c(\tilde{\Lambda}),\ \ G(\tilde{\Lambda}),\ \ S(\tilde{\Lambda})$

1. $\tilde{S} := S$
2. $\texttt{rake} := G$
3. **from** 1 **to** $\tilde{s}$ **do**
4. $\quad \texttt{rake} := \texttt{rake} \gg 1$
5. $\quad \tilde{S} := \tilde{S}\ \&\ \texttt{rake}$
6. **return** $\tilde{c} := c + \tilde{s} + 1,\ \ G \mid (1 \gg \tilde{c} - 2),\ \ (\tilde{S} \ll \tilde{s} + 1) \mid (1\,1\,1 \gg \tilde{c} - 3)$

This is the basic *descending step* for an algorithm that, given a level $\gamma > g(\Lambda)$, computes the number $n_\gamma(\Lambda)$ of descendants of Λ having genus γ. We reproduce two different versions for the algorithm: the first one is based on a *depth first search* exploration of the nodes in the tree, and the second one is recursive.

Input: $\gamma,\ \ G(\Lambda),\ \ S(\Lambda),\ \ c(\Lambda),\ \ g(\Lambda),\ \ \lambda_1$

Output: $n_\gamma(\Lambda)$

1. $g := g(\Lambda) \qquad n := 0$
2. $G[g] := G(\Lambda) \qquad S[g] := S(\Lambda) \qquad \texttt{rake}[g] := G(\Lambda)$
3. $c[g] := c(\Lambda) \qquad m[g] := \lambda_1 \qquad \tilde{s}_{last} := \tilde{s}[g] := 0$
4. **while** $g \neq g(\Lambda) - 1$ **do**
5. $\quad$ **while** $\tilde{s}[g] < m[g]$ **and** $S[g]\ \&\ (1 \gg \tilde{s}[g]) = 0$ **do** $\tilde{s}[g] := \tilde{s}[g] + 1$
6. $\quad$ **if** $\tilde{s}[g] = m[g]$ **then**
7. $\quad\quad g := g - 1$
8. $\quad\quad \tilde{s}_{last} := \tilde{s}[g]$
9. $\quad\quad \tilde{s}[g] := \tilde{s}[g] + 1$
10. $\quad$ **else if** $g = \gamma - 1$ **then**
11. $\quad\quad n := n + 1$
12. $\quad\quad \tilde{s}[g] := \tilde{s}[g] + 1$
13. $\quad\quad$ **else**
14. $\quad\quad\quad c[g+1] := c[g] + \tilde{s}[g] + 1$
15. $\quad\quad\quad$ **if** $\tilde{s}[g] = 0$ **and** $m[g] = c[g]$ **then** $m[g+1] := c[g+1]$
16. $\quad\quad\quad$ **else** $m[g+1] := m[g]$
17. $\quad\quad\quad$ **from** $\tilde{s}_{last} + 1$ **to** $\tilde{s}[g]$ **do**
18. $\quad\quad\quad\quad \texttt{rake}[g] := \texttt{rake}[g] \gg 1$
19. $\quad\quad\quad\quad S[g] := S[g]\ \&\ \texttt{rake}[g]$
20. $\quad\quad\quad S[g+1] := (S[g] \ll \tilde{s}[g] + 1) \mid (1\,1\,1 \gg c[g+1] - 3)$
21. $\quad\quad\quad G[g+1] := G[g] \mid (1 \gg c[g+1] - 2)$
22. $\quad\quad\quad g := g + 1$
23. $\quad\quad\quad \texttt{rake}[g] := G[g]$
24. $\quad\quad\quad \tilde{s}_{last} := \tilde{s}[g] := 0$
25. **return** n

To produce the descendants from a given node of genus g through this algorithm, the *descending step* is not performed each time from scratch as it is given above. Indeed, the *raking* process which must be made for a descendant need only be continued, in the loop at lines 17–19, from the one already executed for the previous siblings, if any. As a consequence, for each possible descendant, that is, for every non-negative value $\tilde{s}[g]$ smaller than the multiplicity $m[g]$, a limited number of operations are required: specifically, those at line 5 are carried out once for every such value, whereas those at lines 18 and 19 are carried out once for every value $\tilde{s}[g]$ up to the last sibling, and those at the remaining lines inside the main loop are carried out at most once for each actual descendant. Thus, the computation of *all* descendants of the node requires $O(m[g])$ operations having time complexity at most $O(\gamma)$. This is why, along with the fact that operations on binary strings are very fast in practice, one should expect the algorithm to perform efficiently when compared to other existing ones, for which either the construction of the data structure representing a generic descendant takes time at least $O(\gamma \log \gamma)$, as in [13], or otherwise the identification of its generators requires at best $O(\gamma)$ basic arithmetic operations, as in the setting given in [6, 7].

Recursive version $n_\gamma(G,\, S,\, c,\, g,\, m)$

1. **if** $g = \gamma$ **then** $n := 1$
2. **else**
3. $\quad n := 0 \qquad \tilde{s}_{last} := 0 \qquad \texttt{rake} := G$
4. $\quad$ **for** $\tilde{s}$ **from** 0 **to** $m-1$ **do**
5. $\quad\quad$ **if** $S \,\&\, (1 \gg \tilde{s}) \neq 0$ **then**
6. $\quad\quad\quad$ **from** $\tilde{s}_{last}+1$ **to** $\tilde{s}$ **do**
7. $\quad\quad\quad\quad \texttt{rake} := \texttt{rake} \gg 1$
8. $\quad\quad\quad\quad S := S \,\&\, \texttt{rake}$
9. $\quad\quad\quad \tilde{s}_{last} := \tilde{s}$
10. $\quad\quad\quad \tilde{c} := c + \tilde{s} + 1$
11. $\quad\quad\quad$ **if** $\tilde{s} = 0$ **and** $m = c$ **then** $\tilde{m} := \tilde{c}$ **else** $\tilde{m} := m$
12. $\quad\quad\quad n := n + n_\gamma\big(G \mid (1 \gg \tilde{c} - 2),\ (S \ll \tilde{s}+1) \mid (1\,1\,1 \gg \tilde{c} - 3),\ \tilde{c},\ g+1,\ \tilde{m}\big)$
13. **return** n

Let us conclude by comparing the running times of the two versions given above for the seeds descending algorithm. We also consider three known algorithms for computing the number of semigroups of a given genus, which we briefly summarize as follows. We use the same notations as in Sections 1 and 2.

- **The *Apéry set* algorithm.** A semigroup Λ can be uniquely identified by its Apéry set [1, 18], which is defined by taking, for each congruence class of integers modulo the multiplicity λ_1, the smallest representative in Λ. Then, this algorithm is based on the fact that the generators of Λ other than λ_1 necessarily lie in its Apéry set: they are the non-zero elements in the set which cannot be written, up to a multiple of λ_1, as the sum of any two other non-zero elements in the set. Thus, although the Apéry set of a

descendant is immediately obtained from the Apéry set of Λ, the efficiency of this method is rather limited.

- **The *generators tracking* algorithm.** It is based on the construction given in [6, 7] for the semigroup tree by keeping track of the set of generators at each descending step. Specifically, the node attached to a semigroup Λ can be encoded by its conductor c, its multiplicity λ_1 and a ternary array which is indexed by $1 \leq i < c + \lambda_1$ and whose value (say 0, 1 or 2) at the i-th entry depends on whether i is a gap of Λ, a generator or a non-gap which is not a generator. The array of a descendant $\tilde{\Lambda} = \Lambda \setminus \{\lambda_s\}$ inherits the values from that of Λ at every entry, except for the one with index $i = \lambda_s$. In the very special case in which Λ is ordinary and $s = 1$, exactly two new entries are required and both correspond to generators of $\tilde{\Lambda}$. Otherwise, $\lambda_s - c + 1$ new entries are required but only the last one may correspond to a generator, according to Lemma 1 in [7].

- **The *decomposition numbers* algorithm.** Fromentin and Hivert consider in [13] the *decomposition numbers*

$$d_\Lambda(x) := \#\{y \in \Lambda \mid x - y \in \Lambda,\ 2y \leq x\}$$

  for $x \in \mathbb{N}$, attached to a semigroup Λ. These numbers, which are intimately related to the extensively used ν-sequence in coding theory [12, 16, 14, 4, 5], are exploited in a simple way in [13] to efficiently explore the semigroup tree.

For each algorithm, we made a first implementation using a *depth first search* (DFS) exploration of the semigroup tree, as in [13], since this consumes much less memory than a *breadth first search* exploration and so allows us to reach further in the computations. We then implemented recursive versions for the algorithms. All implementations were made in plain `C`. The following table displays the number of seconds that each of them required to compute the number of semigroups of genus g for $30 \leq g \leq 40$.

	30	**31**	**32**	**33**	**34**	**35**	**36**	**37**	**38**	**39**	**40**
Apéry - DFS	13	24	39	67	114	193	327	554	933	1577	2657
Apéry - recursive	10	16	28	47	81	136	232	393	634	1071	1805
decomposition - DFS	10	16	27	46	79	131	222	373	626	1050	1762
generators - DFS	8	14	23	39	65	110	185	310	518	868	1448
decomposition - recursive	7	12	20	35	58	97	165	275	462	775	1297
generators - recursive	2	4	7	11	19	31	53	87	145	241	400
seeds - DFS	1	3	4	8	12	21	35	58	96	161	269
seeds - recursive	1	2	3	6	9	15	26	42	70	118	195

References

[1] R. Apéry. Sur les branches superlinéaires des courbes algébriques. *C. R. Acad. Sci. Paris*, 222:1198–1200, 1946.

[2] V. Blanco, P. A. García-Sánchez, and J. Puerto. Counting numerical semigroups with short generating functions. *Internat. J. Algebra Comput.*, 21(7):1217–1235, 2011.

[3] V. Blanco and J. C. Rosales. The set of numerical semigroups of a given genus. *Semigroup Forum*, 85(2):255–267, 2012.

[4] M. Bras-Amorós. Acute semigroups, the order bound on the minimum distance, and the Feng-Rao improvements. *IEEE Trans. Inform. Theory*, 50(6):1282–1289, 2004.

[5] M. Bras-Amorós. A note on numerical semigroups. *IEEE Trans. Inform. Theory*, 53(2):821–823, 2007.

[6] M. Bras-Amorós. Fibonacci-like behavior of the number of numerical semigroups of a given genus. *Semigroup Forum*, 76(2):379–384, 2008.

[7] M. Bras-Amorós. Bounds on the number of numerical semigroups of a given genus. *J. Pure Appl. Algebra*, 213(6):997–1001, 2009.

[8] M. Bras-Amorós. The ordinarization transform of a numerical semigroup and semigroups with a large number of intervals. *J. Pure Appl. Algebra*, 216(11):2507–2518, 2012.

[9] M. Bras-Amorós and S. Bulygin. Towards a better understanding of the semigroup tree. *Semigroup Forum*, 79(3):561–574, 2009.

[10] M. Delgado. Homepage. http://cmup.fc.up.pt/cmup/mdelgado/numbers/.

[11] S. Elizalde. Improved bounds on the number of numerical semigroups of a given genus. *J. Pure Appl. Algebra*, 214(10):1862–1873, 2010.

[12] G. L. Feng and T. R. N. Rao. A simple approach for construction of algebraic-geometric codes from affine plane curves. *IEEE Trans. Inform. Theory*, 40(4):1003–1012, 1994.

[13] J. Fromentin and F. Hivert. Exploring the tree of numerical semigroups. *Mathematics of Computation*, To appear, 2016.

[14] T. Høholdt, J. H. van Lint, and R. Pellikaan. *Algebraic Geometry codes*, pages 871–961. North-Holland, Amsterdam, 1998.

[15] N. Kaplan. Counting numerical semigroups by genus and some cases of a question of Wilf. *J. Pure Appl. Algebra*, 216(5):1016–1032, 2012.

[16] C. Kirfel and R. Pellikaan. The minimum distance of codes in an array coming from telescopic semigroups. *IEEE Trans. Inform. Theory*, 41(6, part 1):1720–1732, 1995. Special issue on algebraic geometry codes.

[17] E. O'Dorney. Degree asymptotics of the numerical semigroup tree. *Semigroup Forum*, 87(3):601–616, 2013.

[18] J. C. Rosales and P. A. García-Sánchez. *Numerical semigroups*, volume 20 of *Developments in Mathematics*. Springer, New York, 2009.

[19] J. C. Rosales, P. A. García-Sánchez, J. I. García-García, and J. A. Jiménez Madrid. The oversemigroups of a numerical semigroup. *Semigroup Forum*, 67(1):145–158, 2003.

[20] J. C. Rosales, P. A. García-Sánchez, J. I. García-García, and J. A. Jiménez Madrid. Fundamental gaps in numerical semigroups. *J. Pure Appl. Algebra*, 189(1-3):301–313, 2004.

[21] N. J. A. Sloane. The on-line encyclopedia of integer sequences.

[22] A. Zhai. Fibonacci-like growth of numerical semigroups of a given genus. *Semigroup Forum*, 86(3):634–662, 2013.

[23] Y. Zhao. Constructing numerical semigroups of a given genus. *Semigroup Forum*, 80(2):242–254, 2010.

Maria Bras-Amorós
maria.bras@urv.cat

Departament d'Enginyeria Informàtica i Matemàtiques
Universitat Rovira i Virgili
Avinguda dels Països Catalans, 26
E-43007 Tarragona

Julio Fernández-González
julio.fernandez.g@upc.edu

Departament de Matemàtiques
Universitat Politècnica de Catalunya
EPSEVG – Avinguda Víctor Balaguer, 1
E-08800 Vilanova i la Geltrú