Maria Bras-Amorós, Julio Fernández-González

The right-generators descendant of a numerical semigroup

Accepted Manuscript

The right-generators descendant
of a numerical semigroup

Maria Bras-Amorós, Julio Fernández-González

November 6, 2019

Abstract

For a numerical semigroup, we encode the set of primitive elements that are larger than its Frobenius number and show how to produce in a fast way the corresponding sets for its children in the semigroup tree. This allows us to present an efficient algorithm for exploring the tree up to a given genus. The algorithm exploits the second nonzero element of a numerical semigroup and the particular *pseudo-ordinary* case in which this element is the conductor.

1   Introduction

Let Λ be a *numerical semigroup*, that is, an additive submonoid with finite complement in the ordered set $\mathbb{N}_0$ of nonnegative integers. The *gaps* of Λ are the elements in the set $\mathbb{N}_0 \setminus \Lambda$, and so the elements of Λ are also called *nongaps*. The number g of gaps is the *genus* of Λ. We assume $g \neq 0$, which amounts to saying that Λ is not the trivial semigroup $\mathbb{N}_0$.

Let c denote the *conductor* of Λ, namely the least upper bound in Λ of the set of gaps. Thus, $c - 1$ is the largest gap of the semigroup, which is known as its *Frobenius number*. We refer to the nonzero nongaps that are smaller than the Frobenius number as *left elements*.

The lowest nonzero nongap m is the *multiplicity* of Λ. We say that Λ is *ordinary* whenever all its gaps are smaller than m, that is, whenever $c = m$. This is also equivalent to asking the genus and the Frobenius number of Λ to be equal. In other words, Λ is ordinary if and only if it has no left elements. Otherwise, the set of left elements of Λ is

$$\Lambda \cap \{m,\, m + 1,\, \ldots,\, c - 2\}$$

and, in particular, it always contains the multiplicity m.

A *primitive element* of Λ is a nongap σ such that $\Lambda \setminus \{\sigma\}$ is still a semigroup, that is, a nonzero nongap that is not the sum of two nonzero nongaps. Primitive elements constitute precisely the minimal generating set of the semigroup. We

use the term *right generators* to refer to those primitive elements that are not left elements. As it comes rightaway from the definitions, Λ has at most m right generators, since they must lie in the set

$$\{c,\, c+1,\, \ldots,\, c+m-1\}.$$

The lowest primitive element of Λ is m. It is a right generator if and only if Λ is ordinary; in this case, Λ has exactly m right generators.

Any numerical semigroup of genus $g \geq 2$ can be uniquely obtained from a semigroup of genus $g-1$ by removing a right generator. This idea gives rise to the construction of the semigroup tree, which was first introduced in [13], [14] and also considered in [3], [4], [5]. The nodes of this tree at level g are all numerical semigroups of genus g. The children, if any, of a given node arise through the removals of each of its right generators. The first levels are shown in Figure 1, where each semigroup is represented by its set of gaps, as in [9].

Figure 1

Lemma 1.1. *Let $\tilde{\Lambda}$ be the child obtained by removing a right generator σ from Λ. Then, the primitive elements of Λ that become right generators of $\tilde{\Lambda}$ are those which are larger than σ.*

Proof. Since $\sigma \geq c$, the semigroup $\tilde{\Lambda}$ has Frobenius number σ. Moreover, any primitive element of Λ other than σ is clearly a primitive element of $\tilde{\Lambda}$ too. $\square$

If follows from Lemma 1.1 that, when successively removing the right generators of Λ in increasing order, one obtains a numerical semigroup at each step. This leads to the main concept around which this paper is built.

Definition 1.2. The *right-generators descendant* (RGD) of Λ is the numerical semigroup $\mathcal{D}(\Lambda)$ obtained by removing from Λ all its right generators.

Notice that $\mathcal{D}(\Lambda)$ is just Λ precisely when Λ does not have any right generators. Otherwise, it is a descendant of Λ in the semigroup tree at some level.

In the next section we show how to compute efficiently from $\mathcal{D}(\Lambda)$ the RGD of the children of Λ. To this end, we restate here a special case of Lemma 2.1 in [6], where right generators correspond to *order-zero seeds*. The result complements Lemma 1.1 by dealing with *new right generators*, that is, with the right generators of a child that are not primitive elements of its parent. The key point goes back to Lemma 3 in [4]. We provide a short proof whose content is partially needed later on.
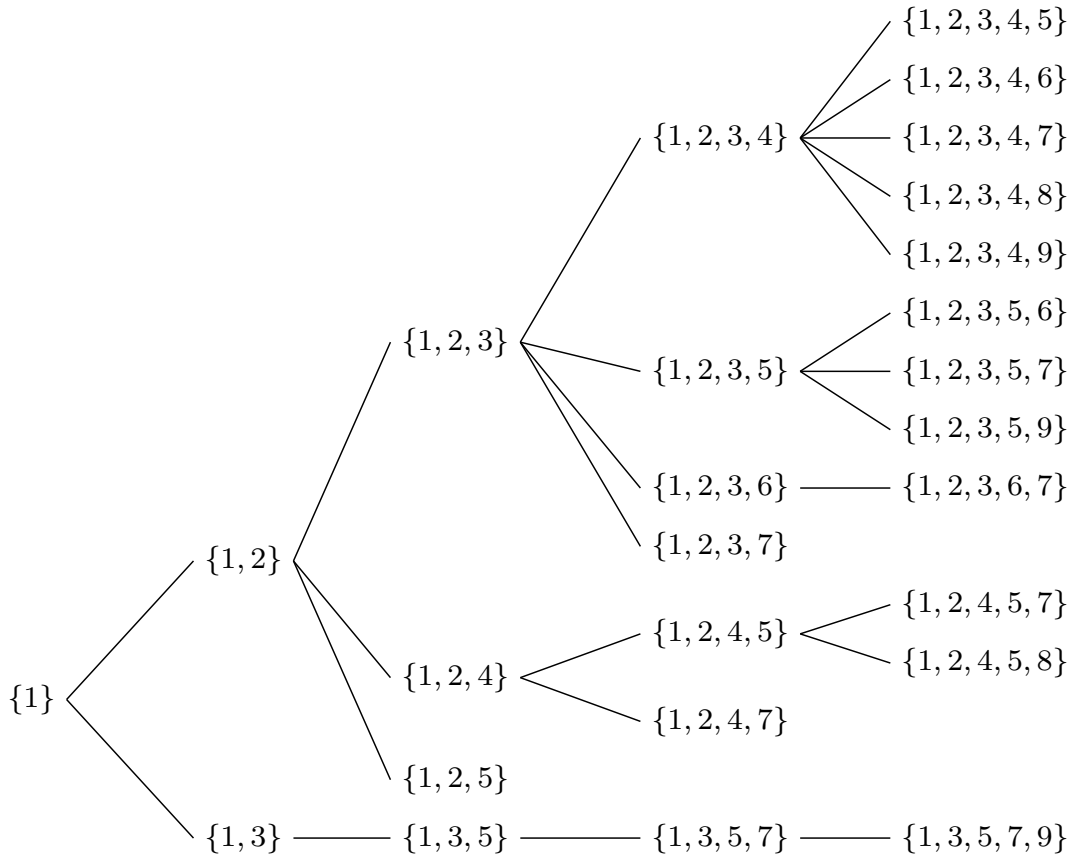
Lemma 1.3. *Let $\tilde{\Lambda}$ be the child obtained by removing a right generator σ from Λ. The semigroup $\tilde{\Lambda}$ is ordinary if and only if Λ is ordinary and $\sigma = m$. Otherwise, the only possible new right generator of $\tilde{\Lambda}$ could be $\sigma + m$.*

Proof. The conductor of $\tilde{\Lambda}$ is $\sigma + 1$. If $\sigma = m$, which can only occur when Λ is ordinary, then $\tilde{\Lambda}$ has multiplicity $m + 1$ and it is also ordinary. If $\sigma \neq m$, then $\tilde{\Lambda}$ has still multiplicity m so it cannot be ordinary. Moreover, new right generators of $\tilde{\Lambda}$ must be of the form $\sigma + \lambda$ for some $\lambda \geq m$, so the result follows from the fact that any primitive element must be smaller than the sum of the conductor and the multiplicity. $\square$

Corollary 1.4. *The semigroup Λ is ordinary if and only if $\mathcal{D}(\Lambda)$ is ordinary. In this case, the conductor of $\mathcal{D}(\Lambda)$ is twice the conductor of Λ.*

Proof. If $\sigma_1, \ldots, \sigma_k$ are the right generators of Λ in increasing order, then Lemma 1.1 allows us to construct $\mathcal{D}(\Lambda)$ through the chain of semigroups

$$\Lambda_0 = \Lambda \supset \Lambda_1 \supset \cdots \cdots \supset \Lambda_k = \mathcal{D}(\Lambda),$$

where Λ_j is obtained by removing σ_j from Λ_{j-1} for $j = 1, \ldots, k$. By Lemma 1.3, Λ_j is ordinary if and only if Λ_{j-1} is ordinary, since σ_j is the lowest right generator of Λ_{j-1}. This implies the first part of the statement. The second part follows from the fact that $c, c+1, \ldots, 2c-1$ are the primitive elements of the ordinary semigroup with conductor c. $\square$

The main goal of this article is an algorithm to explore the semigroup tree which is based on a binary encoding of the RGD. A preliminary version is given in Section 2, and then the algorithm is presented in full form in Section 4. An important role is played by what we call the *jump* of a numerical semigroup, which is the difference between its first two nonzero elements. This parameter is exploited in Section 3 for our purposes, leading to a structure of the algorithm for which the subtrees of semigroups with the same multiplicity and jump can be explored in parallel. In Section 5, we test the efficiency of the algorithm by applying it to the computational problem of counting numerical semigroups by their genus. Our running times are shorter than those required by the best known algorithms.

2 Binary encoding of the RGD

Let us fix a numerical semigroup Λ with multiplicity m and conductor c. We associate with Λ the binary string

$$D(\Lambda) \;=\; D_0 \, D_1 \, \cdots \, D_j \, \cdots$$

encoding its RGD as follows: for $j \geq 0$,

$$D_j := \begin{cases} \mathbf{0} & \text{if } m+j \in \mathcal{D}(\Lambda), \\ \mathbf{1} & \text{otherwise.} \end{cases}$$

The left elements of Λ can be read from the first $c-m$ bits of $D(\Lambda)$. Indeed, for $0 \leq j < c-m$, one has $D_j = \mathbf{0}$ if and only if $m+j$ is not a gap of Λ. In particular, this piece of the string $D(\Lambda)$, together with the multiplicity m, determines uniquely the semigroup Λ.

As for the rest of the string, only the first m bits may take the value $\mathbf{1}$, depending on whether they correspond to right generators of Λ or not: for $j \geq c-m$, one has $D_j = \mathbf{1}$ if and only if $m+j$ is a primitive element of Λ. In particular, Λ is ordinary if and only if $D_0 = \mathbf{1}$. In this case, $D_j = \mathbf{1}$ for $j < m$.

Since $D_j = \mathbf{0}$ for $j \geq c$, we can identify $D(\Lambda)$ with the chain consisting of its first c bits:

$$D(\Lambda) \;=\; D_0 \, D_1 \, \cdots \, D_{c-1}.$$

The binary chains $D(\Lambda)$, for Λ running over the first levels of the semigroup tree, are displayed in Figure 2, where the bits encoding right generators are highlighted in grey. The vertical dashed line at each chain lies just before the entry corresponding to the conductor of the semigroup. The number of bits after that line is equal to the multiplicity.
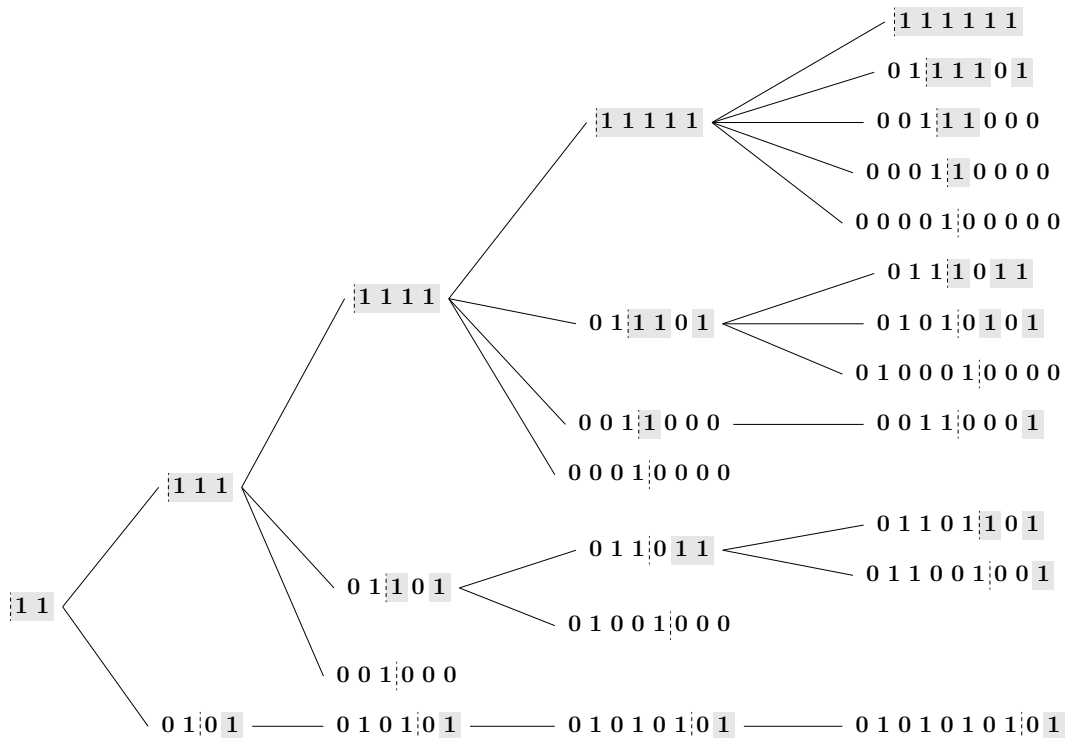

Figure 2

Lemma 2.1. *If Λ is ordinary, then it has exactly m children, namely*

$$\tilde{\Lambda}_s = \Lambda \setminus \{m+s\} \qquad \text{for} \qquad 0 \le s < m.$$

The child $\tilde{\Lambda}_0$ is the ordinary semigroup of multiplicity $m+1$, which means that the bit chain associated with this child is

$$D(\tilde{\Lambda}_0) = \vdots \underbrace{\mathbf{1}\ \mathbf{1}\ \cdots\cdots\ \mathbf{1}}_{m+1}.$$

The other children have multiplicity m, and only $\tilde{\Lambda}_1$ has a new right generator, namely $2m+1$. More precisely, the bit chains associated with them are as follows:

$$D(\tilde{\Lambda}_1) = \mathbf{0}\ \mathbf{1} \vdots \underbrace{\mathbf{1}\ \cdots\ \mathbf{1}}_{m-2}\ \mathbf{0}\ \mathbf{1},$$

whereas, assuming $m > 2$,

$$D(\tilde{\Lambda}_s) = \underbrace{\mathbf{0}\ \cdots\ \mathbf{0}}_{s}\ \mathbf{1} \vdots \underbrace{\mathbf{1}\ \cdots\ \mathbf{1}}_{m-s-1}\underbrace{\mathbf{0}\ \cdots\ \mathbf{0}}_{s+1} \qquad \text{for} \qquad 2 \le s \le m-1.$$

Proof. The right generators of Λ are m, $m+1$, ..., $2m-1$. By Lemma 1.3, $\tilde{\Lambda}_s$ is not ordinary if and only if $s \ne 0$, and then the only possible new right generator of $\tilde{\Lambda}$ could be $2m+s$. Since $\tilde{\Lambda}_1$ has multiplicity m and conductor $m+2$, the integer $2m+1$ is a primitive element of this child. By contrast, $2m+s$ is not a primitive element of $\tilde{\Lambda}_s$ if $s > 1$, because $m+1$ and $m+s-1$ are both nongaps of this child. The result follows then from Lemma 1.1 and the definition of the bit chain encoding the RGD of a numerical semigroup. $\qquad \square$

The children $\tilde{\Lambda}_1, \ldots, \tilde{\Lambda}_{m-1}$ in Lemma 2.1 are the numerical semigroups of multiplicity m having genus m. We refer to them as *quasi-ordinary* semigroups.

Let us go back to the general case and consider a child $\tilde{\Lambda}$, which is obtained by removing a right generator σ from Λ. The index s corresponding to σ in the bit chain $D(\Lambda)$, namely $s = \sigma - m$, satisfies $c - m \le s < c$. The condition $D_s = \mathbf{1}$ is equivalent to σ being a right generator of Λ. Notice that the Frobenius number of $\tilde{\Lambda}$ is $m + s$.

Lemma 2.2. *Let $D(\tilde{\Lambda}) = \tilde{D}_0\,\tilde{D}_1\,\cdots\,\tilde{D}_{m+s}$ be the bit chain associated with $\tilde{\Lambda}$. Then,*

$$\widetilde{D}_j = \begin{cases} D_j & \text{for} \quad 0 \le j < c-m, \\[4pt] \mathbf{0} & \text{for} \quad c-m \le j < s, \\[4pt] D_j & \text{for} \quad s \le j < c, \\[4pt] \mathbf{0} & \text{for} \quad c \le j < m+s. \end{cases}$$

Furthermore, $\tilde{D}_{m+s} = \mathbf{0}$ if and only if there is a positive integer $\ell \le \lfloor s/2 \rfloor$ such that $\tilde{D}_\ell = \tilde{D}_{s-\ell} = \mathbf{0}$. This amounts to saying that $\tilde{\Lambda}$ does not have a new right generator.

Proof. The statements hold if Λ is ordinary, as can be checked from Lemma 2.1. So we can assume $c \neq m$, hence $s \neq 0$ and $\tilde{\Lambda}$ has multiplicity m. The result comes from the definitions of $D(\Lambda)$ and $D(\tilde{\Lambda})$ along with the following remarks. The left elements of $\tilde{\Lambda}$ are those of Λ together with the integers λ such that $c \leq \lambda < m + s$, which yields the values of $\tilde{D}_j$ for $j < s$, whereas $\tilde{D}_s = \mathbf{1} = D_s$ because $m + s$ is a gap of $\tilde{\Lambda}$. As for the right generators of $\tilde{\Lambda}$, according to Lemma 1.1 and Lemma 1.3, they are those which are inherited from Λ, hence above $m + s$ and below $c + m$, and maybe also $2m + s$. This candidate to new right generator is not a primitive element of $\tilde{\Lambda}$ if and only if there is a positive integer ℓ such that $m + \ell$ and $m + s - \ell$ are nongaps of $\tilde{\Lambda}$. Since the midpoint of these two values is $m + s/2$, such an integer can be assumed to satisfy $\ell \leq \lfloor s/2 \rfloor$. Notice that $m + \ell$ and $m + s - \ell$ are below the Frobenius number $m + s$, so they lie in $\tilde{\Lambda}$ if and only if $\tilde{D}_\ell = \tilde{D}_{s-\ell} = \mathbf{0}$. $\qquad\square$

Lemma 2.2 leads to a procedure to explore the semigroup tree using the RGD binary encoding. More precisely, the following recursive function visits the descendants of the semigroup Λ up to a fixed genus γ. The first input item stands for the binary string associated with Λ, and g stands for the genus. If Λ is not ordinary, line 7 can be deleted, and then $\tilde{m}$ should be replaced with m at line 12. An implementation of this procedure runs on the website [1].

RGD − offspring$_\gamma(D, m, c, g)$

---

1. **if** $g < \gamma$ **then**
2. $\qquad \tilde{D} := D$
3. $\qquad \tilde{g} := g + 1$
4. $\qquad$ **for** s **from** $c - m$ **to** $c - 1$ **do**
5. $\qquad\qquad$ **if** $\tilde{D}_s \neq \mathbf{0}$ **then**
6. $\qquad\qquad\qquad$ Frob $:= m + s$
7. $\qquad\qquad\qquad$ **if** $s = 0$ **then** $\tilde{m} := m + 1$ **else** $\tilde{m} := m$
8. $\qquad\qquad\qquad$ $\ell := 1$
9. $\qquad\qquad\qquad$ mid $:= \lfloor s/2 \rfloor$
10. $\qquad\qquad\qquad$ **while** $\ell \leq$ mid **and** $(\tilde{D}_\ell \neq \mathbf{0}$ **or** $\tilde{D}_{s-\ell} \neq \mathbf{0})$ **do** $\ell := \ell + 1$
11. $\qquad\qquad\qquad$ **if** $\ell >$ mid **then** $\tilde{D}_{\text{Frob}} := \mathbf{1}$
12. $\qquad\qquad\qquad$ **RGD − offspring$_\gamma(\tilde{D}, \tilde{m}, \text{Frob} + 1, \tilde{g})$**
13. $\qquad\qquad\qquad$ $\tilde{D}_{\text{Frob}} := \mathbf{0}$
14. $\qquad\qquad\qquad$ $\tilde{D}_s := \mathbf{0}$

When running this function, the most expensive computation for each descendant is at worst the checking loop at line 10. The number of steps in the loop is bounded by half the conductor of the node, hence by the genus. And each step carries out at most five basic operations, which are constant or logarithmic on the genus. This gives the following estimate for the time complexity.

Proposition 2.3. *The number of bit operations to explore the semigroup tree up to genus γ using the* **RGD − offspring$_\gamma$** *function is*

$$O\left(\gamma \log \gamma \sum_{g \leq \gamma} n_g\right),$$

where n_g stands for the number of numerical semigroups of genus g.

3 Pseudo-ordinary semigroups

As in the previous section, let us fix an integer $m \geq 2$, and let Λ stand for a numerical semigroup with multiplicity m. Let then $m + u$ be the second nonzero element of Λ, that is, its first nongap above m. We refer to the positive integer u as the *jump* of Λ. Obviously, it cannot be larger than m, and the conductor c of Λ is at least $m + u$ unless Λ is ordinary, in which case $u = 1$.

Let us first exploit the jump to improve Lemma 1.3. According to the nomenclature introduced in [5] for the nonordinary case, we say that a right generator σ of Λ is a *strong generator* if $\sigma + m$ is a primitive element of the child $\Lambda \setminus \{\sigma\}$. The next result gives a threshold for strong generators. The ordinary case was already treated separately in Lemma 2.1.

Lemma 3.1. *Let $\tilde{\Lambda}$ be the child obtained by removing a right generator σ from Λ. If Λ is not ordinary and $\sigma \geq c + u$, then σ satisfies the following:*

- *It is not a strong generator of Λ.*

- *The only strong generator of $\tilde{\Lambda}$ could be $\sigma + u$, in which case $2u < m$.*

Proof. For an integer $\lambda \geq \sigma$ other than $\sigma + u$, both $\lambda - u$ and $m + u$ are nonzero elements of the set $\tilde{\Lambda} \setminus \{\lambda\}$ because u is positive and

$$m + u \leq c \leq \sigma - u \leq \lambda - u \neq \sigma.$$

Since $\lambda + m$ is the sum of $\lambda - u$ and $m + u$, it cannot be a primitive element of $\tilde{\Lambda} \setminus \{\lambda\}$ whenever this set is a numerical semigroup. The case $\lambda = \sigma$ amounts to the first item of the statement. As for the second, what remains to be proved is the inequality. So let us assume that $\sigma + u$ is a primitive element of $\tilde{\Lambda}$. By the first item and Lemma 1.3, it is then necessarily a right generator of Λ and, in particular, below $c + m$. The hypothesis $\sigma \geq c + u$ implies then $2u < m$. $\square$

Definition 3.2. We say that Λ is *pseudo-ordinary* whenever $m + u$ is precisely the conductor c, that is, whenever $c - 2$ is the genus of Λ. This condition amounts to asking m to be the only left element of Λ. In this case, the jump u is at least 2.

For an integer u with $2 \leq u \leq m$, let us use the notation Λ_u for the pseudo-ordinary semigroup of multiplicity m and jump u. The conductor of Λ_u is $m + u$. The set of right generators of Λ_u is then

$$\{m + u, \, m + u + 1, \, \ldots \ldots, \, 2m + u - 1\} \setminus \{2m\}.$$

In other words, the bit chain associated with Λ_u is

$$D(\Lambda_u) = \mathbf{0} \ \underbrace{\mathbf{1} \cdots \mathbf{1}}_{u-1} \ \vdots \ \underbrace{\mathbf{1} \cdots \mathbf{1}}_{m-u} \ \mathbf{0} \ \underbrace{\mathbf{1} \cdots \mathbf{1}}_{u-1}.$$

Notice that Λ_2 is the only pseudo-ordinary semigroup of multiplicity m that is also quasi-ordinary, that is, a nonordinary child of an ordinary semigroup. Specifically, it is the child $\tilde{\Lambda}_1$ in Lemma 2.1.

The next result complements Lemma 3.1 for pseudo-ordinary semigroups by showing that the first u or $u - 1$ right generators of Λ_u, depending on whether $2u \leq m$ or not, are strong.

Lemma 3.3. *Let $s \neq m$ be an integer satisfying $u \leq s < 2u$. Then, $m + s$ is a strong generator of the pseudo-ordinary semigroup Λ_u.*

Proof. Let $\tilde{\Lambda}_{u,s}$ be the child of Λ_u obtained by removing the right generator $m + s$. Take a nongap of $\tilde{\Lambda}_{u,s}$ above m, that is, an integer λ such that

$$m + u \leq \lambda \neq m + s.$$

Then,

$$m \neq 2m + s - \lambda \leq s + m - u < m + u,$$

hence $2m + s - \lambda$ cannot be a nonzero element of Λ_u. So $2m + s$ is a primitive element of $\tilde{\Lambda}_{u,s}$ above the Frobenius number $m + s$. $\square$

Notice that, with the notation in the proof of Lemma 3.3, one has

$$\tilde{\Lambda}_{u,u} = \Lambda_{u+1} \qquad \text{for} \qquad 2 \leq u \leq m - 1.$$

In particular, the pseudo-ordinary semigroups of multiplicity m are the nodes of a path in the semigroup tree starting at level m and ending at level $2m - 2$:

$$\Lambda_2 \text{———} \Lambda_3 \text{———} \cdots\cdots \text{———} \Lambda_m$$

We now go back again to the general case in order to characterize pseudo-ordinary descendants in the semigroup tree in terms of the monotony of the jump. Let $\tilde{u}$ denote the jump of a child $\tilde{\Lambda}$ of Λ. As shown in Lemma 1.3, $\tilde{\Lambda}$ inherits the same multiplicity m if and only if it is not ordinary, and then the jump $\tilde{u}$ is at most m. If $\tilde{\Lambda}$, hence also Λ, is ordinary, then $\tilde{u} = u = 1$.

Proposition 3.4. *Let $\tilde{\Lambda}$ be the child obtained by removing a right generator σ from Λ. Then, $\tilde{\Lambda}$ is not pseudo-ordinary if and only if $\tilde{u} = u$. More precisely, $\tilde{\Lambda}$ is pseudo-ordinary, and then $\tilde{u} = u + 1$, exactly in one of these two cases:*

· *If Λ is ordinary and $\sigma = m + 1$.*

· *If Λ is pseudo-ordinary and $\sigma = m + u$. In this case, $2 \leq u < m$.*

With the above notation, $\tilde{\Lambda} = \Lambda_2$ in the first item, whereas $\Lambda = \Lambda_u$ and $\tilde{\Lambda} = \Lambda_{u+1}$ in the second one.

Proof. We can assume that $\tilde{\Lambda}$ has multiplicity m. Its second nonzero element is then $m + \tilde{u}$. The parent Λ is obtained from $\tilde{\Lambda}$ by putting the Frobenius number σ back. Thus, if $m + \tilde{u}$ is a left element of $\tilde{\Lambda}$, then it is necessarily equal to $m + u$ because σ is larger. Conversely, assume that $m + \tilde{u}$ is the conductor of $\tilde{\Lambda}$, which implies $\sigma = m + \tilde{u} - 1$ and $\tilde{u} \geq 2$. If $\tilde{u} = 2$, then σ is the only gap of $\tilde{\Lambda}$ larger than m, hence Λ is ordinary. If $\tilde{u} > 2$, then m is still the only left element of Λ and σ is its conductor, hence $\sigma = m + u$ and the result follows. $\square$

Corollary 3.5. *The second nonzero element of Λ is $m + 1$ if and only if Λ is not pseudo-ordinary nor a descendant of a pseudo-ordinary semigroup.*

4   The RGD algorithm

In this section we produce an algorithm to explore the semigroup tree. This is done by putting together the contents that have been developed so far.

Let us start by exploiting the first item of Lemma 3.1 to present a refined version of the recursive procedure below Lemma 2.2. The input corresponds now to a numerical semigroup that is not ordinary nor pseudo-ordinary. In order to shorten the checking loop at line 10 of the $\mathbf{RGD-offspring}_\gamma$ function, as well as the main loop, and hence reduce the number of times that the former is executed, it incorporates as input parameters both the jump u and the number r of right generators. The semigroup is *handled* in a concrete prescribed way: the corresponding parameters may be printed, for instance, or a certain property may be checked. This is the purpose of the **handle** function at the first line.

$\mathbf{RGD}_\gamma(D,\ m,\ u,\ c,\ g,\ r)$

---

1. **handle** $(D,\ m,\ c)$

2. **if** $g < \gamma$ **then**

3. $\quad \tilde{D} := D$

4. $\quad \tilde{r} := r$

5. $\quad$ **for** s **from** $c - m$ **to** $c - m + u - 1$ **do**

6. $\quad\quad$ **if** $\tilde{D}_s \neq \mathbf{0}$ **then**

7. $\quad\quad\quad$ Frob $:= m + s$

8. $\quad\quad\quad$ $\ell := u$

9. $\quad\quad\quad$ mid $:= \lfloor s/2 \rfloor$

10. $\quad\quad\quad$ **while** $\ell \leq$ mid **and** $\left(\tilde{D}_\ell \neq \mathbf{0}$ **or** $\tilde{D}_{s-\ell} \neq \mathbf{0}\right)$ **do** $\ell := \ell + 1$

11. $\quad\quad\quad$ **if** $\ell >$ mid **then**

12. $\quad\quad\quad\quad$ $\tilde{D}_{\text{Frob}} := \mathbf{1}$

13. $\quad\quad\quad\quad$ $\mathbf{RGD}_\gamma\left(\tilde{D},\ m,\ u,\ \text{Frob}+1,\ g+1,\ \tilde{r}\right)$

14. $\quad\quad\quad\quad$ $\tilde{r} := \tilde{r} - 1$

15. $\quad\quad\quad\quad$ $\tilde{D}_{\text{Frob}} := \mathbf{0}$

16. $\quad\quad\quad$ **else then**

17. $\quad\quad\quad\quad$ $\tilde{r} := \tilde{r} - 1$

18. $\quad\quad\quad\quad$ $\mathbf{RGD}_\gamma\left(\tilde{D},\ m,\ u,\ \text{Frob}+1,\ g+1,\ \tilde{r}\right)$

19. $\quad\quad\quad$ $\tilde{D}_s := \mathbf{0}$

20. $\quad$ **while** $\tilde{r} > 0$ **do**

21. $\quad\quad$ $s := s + 1$

22. $\quad\quad$ **if** $\tilde{D}_s \neq \mathbf{0}$ **then**

23. $\quad\quad\quad$ $\tilde{r} := \tilde{r} - 1$

24. $\quad\quad\quad$ $\mathbf{RGD}_\gamma\left(\tilde{D},\ m,\ u,\ m+s+1,\ g+1,\ \tilde{r}\right)$

25. $\quad\quad\quad$ $\tilde{D}_s := \mathbf{0}$

For an integer $m \geq 2$, let $\mathcal{T}_m$ be the subtree of numerical semigroups with multiplicity m, which is rooted at an ordinary semigroup. We present here an algorithm that sequentially explores $\mathcal{T}_2, \ldots, \mathcal{T}_{\gamma+1}$ up to a fixed level γ as detailed in the pseudocode below. The first six lines perform the walk through the path $\mathcal{T}_2$. Then, for $3 \leq m \leq \gamma$, each subtree $\mathcal{T}_m$ is explored by depth first search. The children of every node are visited by lexicographical order of their gaps. Finally, the last three lines of the pseudocode deal with the ordinary semigroup of genus γ.

The binary string at line 8 corresponds to the ordinary semigroup of multiplicity m. The loop at line 12 runs along the path consisting of the pseudo-ordinary semigroups of multiplicity m whose genus is at most γ, except for the last node of the path, which is treated separately at lines 17 to 20.

The loop at line 23 runs over the quasi-ordinary semigroups of multiplicity m, except for the pseudo-ordinary one. Specifically, the $\mathbf{RGD}_\gamma$ function is applied to the children $\tilde{\Lambda}_2, \ldots, \tilde{\Lambda}_{m-2}$ in Lemma 2.1. The binary string associated with $\tilde{\Lambda}_2$ is introduced at line 21. The variable r in the loop stores the number of right generators of the sibling at each step. The child $\tilde{\Lambda}_{m-1}$ does not have any descendants and it is handled at line 27.

RGD algorithm

Input: level $\gamma \geq 2$

---

1. $D := \mathbf{1\ 1\ 0\ 0} \cdots$

2. $\mathbf{handle}\,(D,\ 2,\ 2)$

3. $D_0 := \mathbf{0}$

4. **for** g **from** 2 **to** γ **do**

5. $\qquad D_{2g-1} := \mathbf{1}$

6. $\qquad \mathbf{handle}\,(D,\ 2,\ 2g)$

7. **for** m **from** 3 **to** γ **do**

8. $\qquad D := \underbrace{\mathbf{1} \cdots \mathbf{1}}_{m}\,\mathbf{0\ 0} \cdots$

9. $\qquad \mathbf{handle}\,(D,\ m,\ m)$

10. $\qquad D_0 := \mathbf{0}$

11. $\qquad \texttt{path\_end} := \min\,(m,\ \gamma + 2 - m)$

12. $\qquad$ **for** u **from** 2 **to** $\texttt{path\_end} - 1$ **do**

13. $\qquad\qquad c := m + u$

14. $\qquad\qquad D_{c-1} := \mathbf{1}$

15. $\qquad\qquad \mathbf{handle}\,(D,\ m,\ c)$

16. $\qquad\qquad \mathbf{pseudo}_\gamma(D,\ m,\ u,\ c,\ m-2)$

17. $\qquad c := m + \texttt{path\_end}$

18. $\qquad D_{c-1} := \mathbf{1}$

19. $\qquad \mathbf{handle}\,(D,\ m,\ c)$

20. $\qquad$ **if** $\texttt{path\_end} < \gamma + 2 - m$ **then** $\mathbf{pseudo}_\gamma(D,\ m,\ m,\ c,\ m-1)$

21. $\qquad D := \mathbf{0\ 0}\,\underbrace{\mathbf{1} \cdots \mathbf{1}}_{m-2}\,\mathbf{0\ 0} \cdots$

22. $\qquad$ $r := m - 3$

23.   **for** s **from** 2 **to** $m-2$ **do**

24. $\mathbf{RGD}_\gamma(D,\ m,\ 1,\ m+s+1,\ m,\ r)$

25. $r := r - 1$

26. $D_s := \mathbf{0}$

27. $\mathbf{handle}\,(D,\ m,\ 2m)$

28. $m := \gamma + 1$

29. $D := \underbrace{\mathbf{1}\cdots\mathbf{1}}_{m}\ \mathbf{0}\ \mathbf{0}\ \cdots$

30. $\mathbf{handle}\,(D,\ m,\ m)$

The **pseudo**$_\gamma$ function computes the children of a given pseudo-ordinary semi-group Λ that are not pseudo-ordinary, that is, not fulfilling the second item of Proposition 3.4, and it launches the exploration of their descendants in the tree. The input parameters of Λ include the jump u, which is just the difference $c-m$ in this case. The variable $\tilde{r}$ stores the number of right generators of the siblings. The loop at line 3, which is a rightaway application of Lemma 3.3, runs over the strong generators of Λ, that is, over the children of Λ having a new right generator. By contrast, the loop at line 11 makes use of the first item in Lemma 3.1 to deal with the rest of siblings.

---

pseudo$_\gamma(D,\ m,\ u,\ c,\ \tilde{r})$

---

1. $\tilde{D} := D$

2. $\tilde{D}_u := \mathbf{0}$

3. **for** s **from** $u+1$ **to** $2u-1$ **do**

4.   **if** $s \neq m$ **then**

5. $\mathrm{Frob} := m + s$

6. $\tilde{D}_{\mathrm{Frob}} := \mathbf{1}$

7. $\mathbf{RGD}_\gamma\big(\tilde{D},\ m,\ u,\ \mathrm{Frob}+1,\ c-1,\ \tilde{r}\big)$

8. $\tilde{r} := \tilde{r} - 1$

9. $\tilde{D}_{\mathrm{Frob}} := \mathbf{0}$

10. $\tilde{D}_s := \mathbf{0}$

11. **for** s **from** $2u$ **to** $c-1$ **do**

12.   **if** $s \neq m$ **then**

13. $\tilde{r} := \tilde{r} - 1$

14. $\mathbf{RGD}_\gamma\big(\tilde{D},\ m,\ u,\ m+s+1,\ c-1,\ \tilde{r}\big)$

15. $\tilde{D}_s := \mathbf{0}$

---

5 Counting the numerical semigroups of a given genus

Let n_g stand for the number of numerical semigroups of genus g. It was conjectured in [3] to satisfy $n_{g+2} \geq n_{g+1} + n_g$ and to behave asymptotically like the Fibonacci sequence. The latter was settled in [16]. The inequality has been proved in [9] for the large subset of semigroups whose conductor is not over three times the multiplicity. The weaker conjecture $n_{g+1} \geq n_g$, stated in [2], remains also open for the general case. The development of fast algorithms for computing n_g is motivated to a great extent by these conjectures. See [12] for a nice survey on related results.

We applied the RGD algorithm to the computation of n_γ for a given $\gamma \geq 4$ through an implementation in C. The code is available at [7]. Although it follows the pseudocode in the previous section quite faithfully, some due modifications have been made.

To begin with, the **handle** function is now pointless and must be ignored throughout, because we are not interested in the semigroups themselves but in the amount of them at level γ. In particular, the first six and the last three lines of the pseudocode are omitted: obviously, the subtrees $\mathcal{T}_2$ and $\mathcal{T}_{\gamma+1}$ have both only one node at that level. Also, it follows from Lemma 2.1 that the number of semigroups of genus γ in the subtrees $\mathcal{T}_\gamma$ and $\mathcal{T}_{\gamma-1}$ is, respectively,

$$\gamma - 1 \qquad \text{and} \qquad \gamma - 2 + \sum_{k=1}^{\gamma-4} k.$$

Thus, the last two steps of the loop at line 7 can be skipped by initializing the counter n_γ properly. Moreover, for each multiplicity $3 \leq m < \gamma - 1$, there is only need to explore the subtree $\mathcal{T}_m$ up to level $\gamma - 2$ instead of γ, and then add to n_γ the number of right generators of the children of the nodes at that level. Of course, this remark concerns the implementation of the **RGD**$_\gamma$ function too. In regard to this point, let us also mention the following adjustments and shortcuts in the code:

· In order to reduce the number of comparisons, lines 11 to 20 are rearranged into two possible blocks depending on whether $2m < \gamma$ or not. In the first case, the structure of the block is essentially the same as in the pseudocode: the loop runs up to $u = m - 1$, and then the **pseudo**$_\gamma$ function is called with the parameters at line 20 for $c = 2m$. In the second case, the loop runs up to $u = \gamma - m - 1$, and then we apply the counting that is explained in the next item.

· As a consequence of Lemma 3.1 and Lemma 3.3, the number of grandchildren, say $\mathtt{gc}(\Lambda_u)$, of the pseudo-ordinary semigroup of jump u in $\mathcal{T}_m$ can be easily computed. Indeed, the $m - 1$ children of Λ_u split into two groups: the first u or $u - 1$ siblings, depending on whether $2u \leq m$ or not, do have a new right generator, whereas the others do not. Thus,

$$\mathtt{gc}(\Lambda_u) = \begin{cases} \mathtt{gc}^*(m) + u & \text{if} \quad 2u \leq m, \\ \mathtt{gc}^*(m) + u - 1 & \text{otherwise,} \end{cases}$$

where $\mathtt{gc}^*(m)$ is the sum of the number of right generators of the children

of Λ_u that are inherited from their parent, namely

$$\mathtt{gc}^*(m) \ = \ (m-2) + (m-3) + \cdots\cdots + 1 \ = \ \frac{(m-1)(m-2)}{2} \ .$$

This formula is used in the code for $u = \gamma - m$ whenever this difference is at most m.

· We drop both γ and g as input parameters in the $\mathbf{RGD}_\gamma$ function throughout the recursion, and the difference $\gamma - 2 - g$ is stored and updated instead.

· The last iteration of the loop at line 20 of $\mathbf{RGD}_\gamma$ can be skipped.

The code in C showed that the algorithm turns out to perform faster when the checking loop in the $\mathbf{RGD}_\gamma$ function is implemented in descending order. Thus, lines 8 to 11 are replaced with the following:

$\ell := \lfloor s/2 \rfloor$
$\mathbf{while}\ \ \ell \geq u\ \ \mathbf{and}\ \ \big(\tilde{D}_\ell \neq \mathbf{0}\ \ \mathbf{or}\ \ \tilde{D}_{s-\ell} \neq \mathbf{0}\big)\ \ \mathbf{do}\ \ \ell := \ell - 1$
$\mathbf{if}\ \ \ell < u\ \ \mathbf{then}$

Lastly, we should add that the loops in the $\mathbf{pseudo}_\gamma$ function are rearranged in the implementation to avoid multiple repetitions of the check at lines 4 and 12. Thus, depending on whether $2u \leq m$ or not, one of two possible blocks of three loops is performed instead. The resulting code is less compact but quite more efficient.

The RGD algorithm is based on a much simpler concept than the *seeds algorithm* in [6]. Although the pseudocode is not so short and concise, it is more suitable to implement in any programming language, since it does not require any bitwise operations through long integer data types. Most importantly, it turns out to be computationally much faster. The following table displays the running times in seconds to compute n_γ for $35 \leq \gamma \leq 45$. For the new algorithm, they are more than five times shorter than those which we obtain through a recursive implementation in C of the seeds algorithm, and this improvement grows with the genus. The computations were made on a machine equipped with an Intel® Core™ i7-920 CPU running at 2.67GHz.

γ	35	36	37	38	39	40	41	42	43	44	45
seeds	9.4	15.6	25.9	42.9	71.1	118.1	195.7	323	536	886	1465
RGD	1.7	2.8	4.5	7.2	11.9	19.1	30.8	50	81	131	211

The computational problem of exploring the semigroup tree to produce the sequence of integers n_g is tackled in [11] in a most efficient way. The authors present a depth first search algorithm that is based on the decomposition numbers of a numerical semigroup, and then they optimize it at a highly technical level by using SIMD methods, parallel branch exploration of the tree, and some implementation tricks, such as a partial derecursivation by means of a stack. The source code, which is available at [10], is written in Cilk++ [15], an extension to the C++ language that is designed for multithreaded parallel computing.

We adapted our code to `Cilk++` [8] in order to parallelize the exploration of the semigroup tree. This is performed at a double level that is induced by the very structure of the RGD algorithm. The first level corresponds to the trees $\mathcal{T}_m$ for $m \geq 3$. The second one, to the subtrees of $\mathcal{T}_m$ that are rooted at the pseudo-ordinary semigroups, when removing the edges of the path joining these nodes. The implementation in `C` of the algorithm is adjusted so as to apply the `cilk_for` command to the loops corresponding to lines 7 and 12 of the pseudocode. The following table displays the seconds that were needed to compute n_γ for $45 \leq \gamma \leq 55$ on the same machine as above, but with the eight cores of its CPU running one thread each simultaneously. Notice that the entry for $\gamma = 45$ is now more than four times smaller.

γ	45	46	47	48	49	50	51	52	53	54	55
$\mathbf{RGD}_{8\,t}$	48.0	75.4	126.7	209.6	333.3	564.1	910.6	1441	2320	3876	6481

Finally, we modified the `Cilk++` code of the RGD algorithm so that it produces the same output as the code in [10], namely the list of all integers n_g for genus g up to γ. Then we executed both of them for $45 \leq \gamma \leq 55$ and on eight threads, still on the same machine. The running times in seconds are displayed in the table below. The figures seem to indicate that the improvement grows asymptotically with the genus.

γ	45	46	47	48	49	50	51	52	53	54	55
$\mathbf{F\text{-}H}_{8\,t}$	69.0	112.1	183.0	296.9	483.5	822.5	1339	2175	3536	5755	9948
$\mathbf{RGD}_{8\,t}^{\mathrm{all}}$	56.1	93.2	151.1	249.2	400.3	655.2	1076	1690	2728	4620	7266

Let us complete this section with a computational result that has been obtained by executing the RGD algorithm on a shared server equipped with an `AMD Ryzen 1700X` CPU running at `3.40GHz` on sixteen threads. It goes one step beyond the data compiled at [10].

Theorem 5.1. *There are exactly* 2604033182682582 *numerical semigroups of genus* 71.

References

[1] The Combinatorial Object Server. `http://combos.org/sgroup`.

[2] M. Bras-Amorós. *On Numerical Semigroups and Their Applications to Algebraic Geometry Codes*, in Thematic Seminar "Algebraic Geometry, Coding and Computing", Universidad de Valladolid en Segovia. `www.singacom.uva.es/oldsite/seminarios/WorkshopSG/workshop2/Bras_SG_2007.pdf`, 2007.

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

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

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

[6] M. Bras-Amorós and J. Fernández-González. Computation of numerical semigroups by means of seeds. *Math. Comp.*, 87(313):2539–2550, 2018.

[7] M. Bras-Amorós and J. Fernández-González. Implementation of the RGD algorithm in `C`. `http:\\crises-deim.urv.cat/~mbras/RGD.cpp`, 2019.

[8] M. Bras-Amorós and J. Fernández-González. Implementation of the RGD algorithm in `Cilk++`. `http:\\crises-deim.urv.cat/~mbras/RGD_cilk.cpp`, 2019.

[9] S. Eliahou and J. Fromentin. Gapsets and numerical semigroups. Preprint available at `https://arxiv.org/format/1811.10295`.

[10] J. Fromentin and F. Hivert. `https://github.com/hivert/NumericMonoid`.

[11] J. Fromentin and F. Hivert. Exploring the tree of numerical semigroups. *Math. Comp.*, 85(301):2553–2568, 2016.

[12] N. Kaplan. Counting numerical semigroups. *Amer. Math. Monthly*, 124(9):862–875, 2017.

[13] 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.

[14] 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.

[15] Software.intel.com. Intel® Cilk™ Plus homepage: `https://www.cilkplus.org/`.

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

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