Exploring the tree of numerical semigroups

Jean Fromentin

EXPLORING THE TREE OF NUMERICAL SEMIGROUPS

JEAN FROMENTIN

Abstract. In this paper we describe an algorithm visiting all the numerical semigroups up to a given genus using a new representation of numerical semigroups.

1. Introduction

A *numerical semigroup* S is a subset of $\mathbb{N}$ containing 0, close under addition and of finite complement in $\mathbb{N}$. For example the set

$$S_E = \{0, 3, 6, 7, 9, 10\} \cup [12, +\infty[\tag{1}$$

is a numerical semigroup. The *genus* of a numerical semigroup S, denoted by $g(S)$, is the cardinality of $\mathbb{N} \setminus S$, *i.e*, $g(S) = \text{card}(\mathbb{N} \setminus S)$. For example the genus of S_E defined in (1) is 6, the cardinality of $\{1, 2, 4, 5, 8, 11\}$

For a given positive integer g, the number of numerical semigroups is finite and is denoted by n_g. In J.A. Sloane's *on-line encyclopedia of integer* [1] we find the values of n_g for $g \leqslant 52$. These values have been obtain by M. Bras-Amorós (view [2] for more details for $g \leqslant 50$). On his home page [3], M. Delgado gives the value of n_{55} without specifying the values of n_{53} and n_{54}. M. -Bras Amorós used a depth first search exploration of the tree of numerical semigroups up to a given genus. This tree was introduced by J.C. Rosales and al. in [4] and it is the subject of the Section 2.

Starting with all the numerical semigroups of genus 49 she obtained the number of numerical semigroups of genus 50 in 18 days on a pentium D runing at 3GHz. In the package `NumericalSgs` [5] of `GAP` [6], M. Delgado together with P.A. Garcia-Sanchez and J. Morais used the same method of exploration.

The paper is divided as follows. In section 2 we describe the tree of numerical semigroups and give bounds for some parameters attached to a numerical semigroup. In Section 3 we describe a new representation of numerical semigroups that is well suited to the construction of the tree. In Section 4 we describe an algorithm based on the representation given in Section 3 and give its complexity. Section 5 is more technical, and is devoted to the optimisation of the algorithm introduced in Section 4.

2. The tree of numerical semigroups

We first start by some notations.

Definition 2.1. Let S be a numerical semigroup. We define
i) $m(S) = \min(S \setminus \{0\})$, the *multiplicity* of S;
ii) $g(S) = \text{card}(\mathbb{N} \setminus S)$, the *genus* of S;
iii) $c(S) = 1 + \max(\mathbb{N} \setminus S)$, the *conductor* of S for S different from $\mathbb{N}$. By convention the conductor of $\mathbb{N}$ is 0.

By definition a numerical semigroup is an infinite object. We need a finite description of such a semigroup. That is the role of generating sets.

Definition 2.2. A subset X of a semigroup is a *generating set* of S if every element of S can be express as a sum of elements in X.

Notation 2.3. A numerical semigroup admitting $X = \{x_1 < x_2 < ... < x_n\}$ as generating set is denoted by $S = \langle x_1, ..., x_\ell \rangle$.

We now introduce a specific generating set.

Definition 2.4. A non-zero element x of a numerical semigroup S is said to be *irreducible* if it cannot be expressed as a sum of two non-zeros elements of S. We denote by $\text{Irr}(S)$ the set of all irreducible elements of S.

Proposition 2.5. *Let S be a numerical semigroup. Then* $\text{Irr}(S)$ *is the minimal generating set of S.*

Proof. Assume for a contradiction, that there exists an integer x in S than cannot be decomposed as a sum of irreducible elements. We may assume that x is minimal with this property. As x cannot be irreducible, there exist y and z in $S \setminus \{0\}$ satisfying $x = y + z$. Since we have $y < x$ and $z < x$, the integers y and z can be expressed as a sum of irreducible elements of S and so $x = y + z$ is a sum of irreducible elements, in contradiction to hypothesis.

As irreducible elements of S cannot be decomposed as the sum of two non-zeros integers in S, they must occur in each generating set of S. □

We recall that Apéry elements of a numerical semigroup S associated to $m(S)$ are the integers x in S such that $x - m(S)$ is no longer in S. We denote by $\text{App}(S)$ the set of these elements. It is well known that the cardinality of $\text{App}(S)$ is exactly $m(S)$ (see [7] for example). Note that the set $\text{Irr}(S)$ is in included in $\text{App}(S)$. In particular, the set $\text{Irr}(S)$ is finite and its cardinality is at most $m(S)$.

If we reconsider the numerical semigroup of (1), we obtain

$$S_E = \{0, 3, 6, 7, 9, 10\} \cup [12, +\infty[= \langle 3, 7 \rangle \tag{2}$$

Let S be a numerical semigroup. The set $T = S \cup \{c(S) - 1\}$ is also a numerical semigroup and its genus is $g(S) - 1$. As $[c(S) - 1, +\infty[$ is included in T we have $c(T) \leqslant c(S) - 1$. Therefore every semigroup S of genus g can be obtained from a semigroup T of genus $g - 1$ by removing an element of T greater than or equal to $c(T)$.

Proposition 2.6. *Let S be a numerical semigroup and x an element of S. The set $S^x = S \setminus \{x\}$ is a numerical semigroup if and only if x is irreducible in S.*

Proof. If x is not irreducible in S, then there exist a and b in $S \setminus 0$ such that $x = a+b$. Since $a \neq 0$ and $b \neq 0$ hold, the integers a and b belong to $S \setminus \{x\}$. Since $x = a + b$ and $x \notin S^x$, it follows that S^x is not stable under addition.

Conversely, assume that x is irreducible in S. As 0 is never irreducible, the set S^x contains 0. Let a and b be two integers belonging to S^x. The set S is stable under addition, hence $a + b$ lies in S. As S is equal to $S^x \cup \{x\}$, the integer $a + b$ also lies in S^x except if it is equal to x. The latter is impossible since a and b are different from x and x is irreducible. □

Proposition 2.6 implies that every semigroup T of genus g can be obtained from a semigroup S by removing a generator x of S that is greater than $c(S)$. Hence relations $T = S^x = S \setminus \{x\}$ hold.

We construct the tree of numerical semigroups, denoted by $\mathcal{T}$ as follows. The root of the tree is the unique semigroup of genus 0, *i.e*, $\langle 1\rangle$ that is equal to $\mathbb{N}$. If S is a semigroup in the tree, the sons of S are exactly the semigroup S^x where x belongs to $\mathrm{Irr}(S) \cap [c(s), +\infty[$. By convention, when depicting the tree, the ,numerical semigroup S^x is in the left of S^y if x is smaller than y.

The above remarks imply that a semigroup S has depth g in $mathcalT$ if and only if its genus is g, see Figure 1. We denote by $\mathcal{T}^{\leqslant g}$ the subtree of $\mathcal{T}$ resticted to all semigroup of genus $\leqslant g$.

As the reader can check, the main difficulty to characterize the sond of a semigroup is to determine its irreducible elements. In [5], the semigroup are given by their Apéry set $\mathrm{App}(S)$ and then the main difficulty is to describe $\mathrm{App}(S_x)$ from $\mathrm{App}(S)$. This approach is elegant but not sufficiently *basic* for our optimisations.

We conclude this section with some basic results on numerical semigroups of a given genus. Let S be a numerical semigroup. We first prove :

$$x \in \mathrm{Irr}(S) \quad \text{implies} \quad x \leqslant c(S) + m(S). \tag{3}$$

If y is a positive integer with $y \geqslant c(S) + m(S)$ then y lies in S (as $y \geqslant c(S)$ holds). Moreover, we always have $y - m(S) \geqslant c(s)$ and so y is not irreducible.

As $\mathbb{N} \setminus S$ contains $\{1, .., m(S) - 1\}$ we have

$$m(S) \leqslant g(S) + 1. \tag{4}$$

Let x be an element of $S \cap [0, c(S) - 1]$. The integer $y = c(S) - 1 - x$ lies in $[0, c(S) - 1]$. Moreover y is not in S, for otherwise we write $c(S) - 1 = y + x$ where x, y are elements of S implying $c(S) - 1 \in S$. In contradiction with the definition of conductor. Thus we define an involution

$$\begin{array}{rcl} \psi : [0, c(S) - 1] & \to & [0, c(S) - 1] \\ x & \mapsto & c(S) - 1 - x \end{array}$$

that send $S \cap [0, c(S) - 1]$ into $[0, c(S) - 1] \setminus S$. The cardinality of $[0, c(S)[\setminus S$ is exactly $g(S)$. Let us denote by k the cardinality of $S \cap [0, c(S) - 1]$. Since ψ is injective we must have $k \leqslant g(S)$. From the relation

$$[0, c(S)[= S \cap [0, c(S)[\sqcup [0, c(S)[\setminus S$$

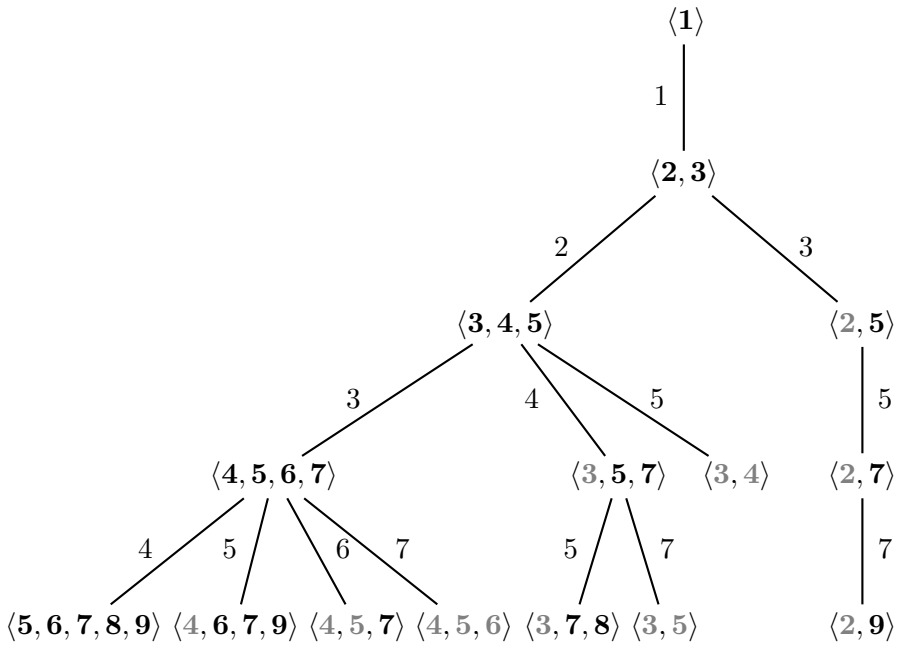

FIGURE 1. The first five layers of the tree $\mathcal{T}$ of numerical semigroups, corresponding to $\mathcal{T}^{\leqslant 5}$. A generator of a semigroup is in gray if it is not in $[c(S), +\infty[$. An edge between a semigroup S and its son S' is labelled by x is S' is obtained from S by removing x, that is $S' = S^x$.

we obtain $c(S) = k + g(S)$ and so

$$c(S) \leqslant 2g(S). \tag{5}$$

3. Decomposition number

The aim of this section is to describe a new representation of numerical semigroups. which is well suited to a efficient exploration of the tree $\mathcal{T}$ of numerical semigroups.

Definition 3.1. Let S be a numerical semigroup. For every x of $\mathbb{N}$ we set

$$D_S(x) = \{y \in S \mid x - y \in S \text{ and } 2y \leqslant x\}$$

and $d_S(x) = \operatorname{card} D_S(x)$. We called $d_S(x)$ the *S-decomposition number* of x. The application $d_S : \mathbb{N} \to \mathbb{N}$ is the *S-decomposition numbers function*.

Assume that y is an element of $D_S(x)$. By very definition of $D_S(x)$, the integer $z = x - y$ also belongs to S. Then x can be decomposed as $x = y + z$ with y and z in S. Moreover the condition $2y \leqslant x$ implies $y \leqslant z$. In other words if we define $D'_S(x)$ to be the set of all $(y, z) \in S \times S$ with $x = y + z$ and $y \leqslant z$ then $D_S(x)$ is the image of $D'_S(x)$ under the projection on the

first coordinate. Hence $D_S(x)$ describes how x can be decomposed as sums of two elements of S,-. This is why $d_S(x)$ is called the S-decomposition number of x.

Example 3.2. Reconsider the semigroup S_E given at (1). The integer 14 admits three decompositions as sums of two elements of S, namely $14 = 0+14$, $14 = 3+11$ and $14 = 7+7$. Thus the set $D_{S_E}(14)$ is equal to $\{0, 3, 7\}$ and d_{S_E} equals 3.

Lemma 3.3. For every numerical semigroup S and every integer $x \in \mathbb{N}$, we have $d_S(x) \leqslant 1 + \left\lfloor \frac{x}{2} \right\rfloor$ and the equality holds for $S = \mathbb{N}$.

Proof. As the set $D_S(x)$ is included in $\{0, ..., \lfloor \frac{x}{2} \rfloor\}$, the relation $d_S(x) \leqslant 1 + \lfloor \frac{x}{2} \rfloor$ holds. For $S = \mathbb{N}$ we have the equality for $D_S(x)$ and so for $d_S(x)$. □

Proposition 3.4. *Let S be a numerical semigroup and $x \in \mathbb{N} \setminus \{0\}$. We have:*

i) x lies in S if and only if $d_S(x) > 0$.

ii) x is in $\mathrm{Irr}(S)$ if and only if $d_S(x) = 1$.

Proof. We start with *i)*. If x is an element of S then x equals $0 + x$. The relation $2 \times 0 \leqslant x$ and $0 \in S$ imply that $D_S(x)$ contains 0, and so $d_S(x) > 0$ holds. Conversely, the relation $d_S(x) > 0$ implies that $D_S(x)$ is non-empty. Let y be an element of $D_S(x)$. As y and $x - y$ belong to S, by definition, the integer $x = (x - y) + y$ is in S (since S is stable by addition).

Let us show *ii)*. Assume x is irreducible in S. There cannot exist y and z in $(S \setminus \{0\})^2$ such that $x = y + z$. The only possible decomposition of x as a sum of two elements of S is $x = 0 + x$. Hence, the set $D_S(x)$ is equal to $\{0\}$ and we have $d_S(x) = 1$. Conversely, let x such that $d_S(x) = 1$. By *i)* the integer x must be in S. As $x = 0 + x$ is always a decomposition of x as a sum of two elements in S, we obtain $D_S(x) = \{0\}$. If there exist y and z in S such that $y \leqslant z$ and $x = y + z$ hold then y lies in $D_S(x)$. This implies $y = 0$ and $z = x$. Hence x is irreducible in S. □

We note that 0 is never irreducible despite the fact $d_S(0)$ is 1 for all numerical semigroup S.

We now explain how to compute the S-decomposition numbers function of a numerical semigroup from these of its father.

Proposition 3.5. *Let S be a numerical semigroup and x be an irreducible element of S. Then for all $y \in \mathbb{N} \setminus \{0\}$ we have*

$$d_{S^x}(y) = \begin{cases} d_S(y) - 1 & \text{if } y \geqslant x \text{ and } d_S(y - x) > 0, \\ d_S(y) & \text{otherwise.} \end{cases}$$

Proof. Let y be in $\mathbb{N} \setminus \{0\}$. We have

$$D_{S^x}(y) = \{z \in S^x \mid y - z \in S^x \text{ and } 2z \leqslant y\}$$

and $D_{S^x}(y)$ is a subset of $D_S(y)$. We have $D_S(y) \setminus D_{S^x}(y) = E \sqcup F$ where

$$E = \begin{cases} \{x\} & \text{if } y - x \in S^x \text{ and } 2x \leqslant y, \\ \emptyset & \text{otherwise.} \end{cases}$$
$$F = \{z \in S \mid y - z = x \text{ and } 2z \leqslant y\}$$

Since the relation $y - x \in S^x$ can be rewritten as the conjunction of $y - x \in S$ and $y \neq 2x$, we obtain

$$E = \begin{cases} \{x\} & \text{if } y - x \in S \text{ and } 2x < y, \\ \emptyset & \text{otherwise.} \end{cases}$$

By Proposition 3.4, the relation $y - x \in S$ is equivalent to $y \geqslant x$ and $d_S(y - x) > 0$, we have

$$E = \begin{cases} \{x\} & \text{if } y \geqslant x \text{ and } d_S(y - x) > 0 \text{ and } 2x < y, \\ \emptyset & \text{otherwise.} \end{cases}$$

On the other hand, we have

$$\begin{aligned} F &= \{z \in S \mid y - z = x \text{ and } 2z \leqslant y\} \\ &= \{z \in S \mid z = y - x \text{ and } y \leqslant 2x\} \\ &= \begin{cases} \{y - x\} & \text{if } y - x \in S \text{ and } y \leqslant 2x, \\ \emptyset & \text{otherwise.} \end{cases} \end{aligned}$$

As in the case of E, we get

$$F = \begin{cases} \{y - x\} & \text{if } y \geqslant x \text{ and } d_S(y - x) > 0 \text{ and } y \leqslant 2x, \\ \emptyset & \text{otherwise.} \end{cases}$$

In E we have the constraint $2x < y$ while in F we have $y \leqslant 2x$, hence only one of the sets E or F can be non-empty. This implies

$$D_S(y) \setminus D_{S^x}(y) = \begin{cases} \{x\} & \text{if } y \geqslant x \text{ and } d_S(y - x) > 0 \text{ and } y \leqslant 2x, \\ \{y - x\} & \text{if } y \geqslant x \text{ and } d_S(y - x) > 0 \text{ and } y > 2x, \\ \emptyset & \text{otherwise.} \end{cases}$$

Therefore we obtain

$$d_{S^x}(y) = \begin{cases} d_S(y) - 1 & \text{if } y \geqslant x \text{ and } d_S(y - x) > 0, \\ d_S(y) & \text{otherwise.} \end{cases}$$

□

4. A new algorithm

We can easily explore the tree of numerical semigroups up to a genus G using a depth search first algorithm using a stack (see 1). This approach does not seem to have been used before. In particular, M. Bras-Amorós and M. Delgado use instead a breadth search first exploration. The main

advantage in our approach is the small memory needs. The cost to pay is that, if we want to explore the tree deepero, we must restart from the root.

Algorithm 1 Depth search first exploration of the tree of numerical semigroups

```
1: procedure EXPLORE(G)
2:     Stack stack                                   ▷ the empty stack
3:     S ← ⟨1⟩
4:     while stack is not empty do
5:         S ← stack.top()
6:         stack.pop()
7:         if g(S) < G then
8:             for x from c(S) to c(S) + m(S) do
9:                 if x ∈ Irr(S) then
10:                    S.push(S^x)
11:                end if
12:            end for
13:        end if
14:    end while
15: end procedure
```

In Algorithm 1 we do not specify how to compute $c(S)$, $g(S)$ and $m(S)$ from S neither how to test if an integer is irreducible. It also miss the characterisation of S^x from S. These items depend heavily of the representation of S. Our choice is to use the S-decomposition numbers function. The first task is to use a finite set of such numbers to characterise the whole semigroup.

Proposition 4.1. *Let G be an integer and S be a numerical semigroup of genus $g \leqslant G$. Then S is entirely described by $\delta_S = (d_S(0), ..., d_S(3\,G))$. More precisely we can obtain $c(S), g(S), m(S)$ and $\mathrm{Irr}(S)$ from δ_S.*

Proof. By (5) we have $c(S) \leqslant g(S)$ and so the S-decomposition number of $c(S)$ occurs in δ_S. Since $c(S)$ is equal to $\max(\mathbb{N} \setminus S)$, Proposition 4.1 implies

$$c(S) = 1 + \max\{i \in [0, ..., 3G],\ d_S(i) = 0\}.$$

As all elements of $\mathbb{N}\setminus S$ are smaller than $c(S)$, their S-decomposition numbers are in δ_S and we obtain

$$g(S) = \mathrm{card}\{i \in [0, .., 3G],\ d_S(i) = 0\}.$$

By (4) the relation $m(S) \leqslant g(S) + 1$ holds. This implies that the S-decomposition number of $m(S)$ appears in δ_S :

$$m(S) = \min\{i \in [0, ..., 3G],\ d_S(i) > 0\}.$$

Since, by (3), all irreducible elements are smaller than $c(S) + m(S) - 1$, which is itself smaller than $3G$, equations (4) and (5) give

$$\mathrm{Irr}(S) = \{i \in [0, ..., 3G],\ d_S(i) = 1\}.$$

□

Even though it is quite simple, the computation of $c(S), m(S)$ and $g(S)$ from δ_S has a non negligible cost. We represent a numerical semigroup S of genus $g \leqslant G$ by $(c(S), g(S), c(S), \delta_S)$. In an algorithmic context, if the variable `S` stands for a numerical semigroup we use:

– `S.c`, `S.g` and `S.m` for the integers $c(S)$, $g(S)$ and $m(S)$;

– `S.d[i]` for the integer $d_S(i)$.

For example the following Algorithm initializes a representation of the semigroup $\mathbb{N}$ ready for an exploration up to genus G.

Algorithm 2 Return the root of $\mathcal{T}$ for an exploration up to genus `G`

```
function ROOT(G)
    R.c ← 1                                   ▷ R stands for ℕ
    R.g ← 0
    R.m ← 1
    for x from 1 to 3G do
        R.d[x] ← 1 + ⌊x/2⌋
    end for
    return R
end function
```

We can now describe an algorithm that returns the representation of the semigroup S^x from that of the semigroup S where x is an irreducible element of S greater than $c(S)$.

Algorithm 3 Returns the son S_x of S with $x \in \mathrm{Irr}(S) \cap [c(S), c(S)+m(S)[$.

```
1:  function SON(S,x,G)
2:      T.c ← x + 1                           ▷ T stands for S_x
3:      T.g ← S.g + 1
4:      if x > S.m then
5:          T.M ← S.m
6:      else
7:          T.M ← S.m + 1
8:      end if
9:      T.d ← S.d
10:     for y from x to 3G do
11:         if S.d[y − x] > 0 then
12:             T.d[y] ← T.d[y] − 1
13:         end if
14:     end for
15:     return T
16: end function
```

Proposition 4.2. *Running on (S, x, G) with $g(S) \leqslant G$, $x \in \mathrm{Irr}(S)$ and $x \leqslant c(S)$, Algorithm 3 returns the numerical semigroup S_x in time $O(\log(G) \times G)$.*

Proof. By construction S^x is the semigroup $S \setminus \{x\}$. Thus the genus S^x is $g(S) + 1$, see Line `1`. Every integer of $I = [x + 1, +\infty[$ lies in S since x is greater than $c(S)$, so the interval I is included in S^x. As x does not belong to S, the conductor of S^x is $x+1$, see Line `2`. For the multiplicity of S^x we have two cases. First, if $x > m(S)$ holds then $m(S)$ is also in S^x and so $m(S^x)$ is equal to $m(S)$. Assume now $x = m(S)$. The relation $x(S) \geqslant c(S)$ and the characterisation of $m(S)$ implies $x = m(S) = c(S)$. Thus S^x contains $m(S) + 1$ which is $m(S^x)$. The initialisation of $m(S^x)$ is done by Lines `4` to `8`. The correctness of the computation of δ_{S^x} (see Proposition 4.1) done from Line `9` to Line `15` is a direct consequence of Proposition 3.4.

Let us now prove the complexity statement. Since by (5) and (4) we have $x \leqslant 3G$ together with $m(S) \leqslant G + 1$, each line from `2` to `8` is done in time $O(\log(G))$. The **for** loop needs $O(G)$ steps and each step is done in time $O(\log(G))$. Summarizing, these results give that the algorithm runs in time $O(\log(G) \times G)$. □

Algorithm 4 Returns an array containing the value of n_g for $g \leqslant G$

```
1:  function COUNT(G)
2:      n ← [0, ..., 0]                ▷ n[g] stands for n_g and is initialised to 0
3:      Stack stack                                          ▷ the empty stack
4:      S ← ROOT(G)
5:      while stack is not empty do
6:          S ← stack.top()
7:          stack.pop()
8:          n[S.g] ← n[S.g] + 1
9:          if S.g < G then
10:             for x from S.c to S.c + S.m do
11:                 if S.d[x] = 1 then
12:                     S.push(SON(S, x, G))
13:                 end if
14:             end for
15:         end if
16:     end while
17:     return n
18: end function
```

Proposition 4.3. *Running on $G \in \mathbb{N}$, Algorithm* COUNT *returns the array $[n_0, ..., n_G]$ in time*

$$O\left(\log(G) \times G \times \sum_{g=0}^{G} n_g\right)$$

and its space complexity is $O(\log(G) \times G^3)$.

Proof. The correctness of the algorithm is a consequence of Proposition 4.2 and of the description of the tree $\mathcal{T}$ of numerical semigroups.

For the time complexity, let us remark that Algorithm Son is called for every semigroup of the tree $\mathcal{T}^{\leqslant G}$ (the restriction of $\mathcal{T}$ to semigroup of genus $\leqslant G$). Since there are exactly $N = \sum_{g=0}^{G} n_g$ such semigroups, the time complexity of Son established in Proposition 4.2 guarantees that the running time of Count is in $O(\log(G) \times G \times N)$, as stated.

Let us now prove the space complexity statement. For this we need to describe the stack through the run of the algorithm. Since the stack is filled with a depth first search algorithm, it has two properties. The first one is that reading the stack from the bottom to the top, the genus of semigroup increases. The second one is that, for all $g \in [0, G]$, every semigroup of genus g in the stack has the same father. As the number of sons of a semigroup S is the number of S-irreducible elements in $[c(S), c(S)+m(S)-1]$, a semigroup S has at most $m(S)$ sons. By (4), this implies that a semigroup of genus g as at most $g+1$ sons. Therefore the stack contains at most $g+1$ semigroup of genus $g+1$ for $g \leqslant G$. So the size of the stack is bounded by

$$S = \sum_{g=0}^{G} g = \frac{G(G+1)}{2}$$

A semigroup is represented by a quadruple $(c(S), g(S), m(S), \delta_S)$. By equations (5) and (4), we have $c \leqslant 2g(S)$ and $m \leqslant g(S)+1$. As $g(S) \leqslant G$ holds, the integers c, g and m of the representation of S require a memory space in $O(\log(G))$. The size of $\delta_S = (d_S(0), ..., d_S(3G))$ is exactly $3G+1$. Each entry of δ_S is the S-decomposition number of an integer smaller than $3G$ and hence requires $\log(G)$ bytes of memory space. Therefore the space complexity of δ_S is in $O(\log(G) \times G)$, which implies that the space complexity of the Count algorithm is

$$O(\log(G) \times G \times S) = O(\log(G) \times G^3).$$

□

5. Technical optimizations and results

Assume for example that we want to construct the tree $\mathcal{T}^{\leqslant 100}$ of all numerical semigroup of genus smaller than 100. In this case, the representation of numerical semigroup given in Section 3 uses decomposition numbers of integers smaller than 300. By Lemma 3.3, such a decomposition number is smaller than 151 and requires 1 byte of memory. Thus at each **for** step of Algorithm Son, the CPU actually works on 1 byte. However current CPUs work on 8 bytes. The first optimization uses this point.

To go further we must specify that the array of decomposition numbers in the representation of a semigroup corresponds to consecutive bytes in memory. In the **for** loop of Algorithm Son we may imagine two cursors:

the first one, denoted `src` pointing to the memory byte of `S.d[0]` and the second one, denoted `dst` pointing to the memory byte `T.d[y]`. Using these two cursors, Lines `10` to `14` of Algorithm Son can be rewritten as follows

```
src ← address(S.d[0])
dst ← address(T.d[x])
i ← 0
while i ⩽ 3G − x do
    if content(src) > 0 then
        decrease content(dst) by 1
    end if
    increase src,dst,i by 1
end while
```

In this version we can see that the cursors `src` and `dst` move at the same time and that the modification of the value pointed by `dst` only needs to access the values pointed by `src` and `dst`. We can therefore work in multiple entries at the same time without collision. Current CPUs allow this thanks to the SIMD technologies as MMX, SSE, etc. The acronym SIMD stands for Single Operation Multiple Data.

The MMX technology permits to work on 8 bytes in parallel while the SSE works in 16 bytes. As the rest of the CPU works on 8 bytes, the SSE technology needs some constraint on memory access than cannot be fulfilled in our algorithm. This motivate our choice to use the MMX technology. More precisely, we use three commands: `pcmpeqb`, `pandn` and `psubb`. These commands work on two arrays of 8 bytes, called `s` and `d` here. In each case the array `d` is modified. We denote bytes by $\{0,1\}$-words of length 8. After a call to `pcmpeqb(s,d)` the ith entry of d contains `11111111` if $\mathtt{s}[i] = \mathtt{d}[i]$ holds and `00000000` otherwise. The command `pand(s,d)` store in $\mathtt{d}[i]$ the value of the byte-logic operation `s`[i] `and` (`not` `d`[i]). When `pand(s,d)` is called the new value of $\mathtt{d}[i]$ is $\mathtt{s}[i] - \mathtt{d}[i]$. Glueing all the pieces together we obtain the following version of the **for** loop of Algorithm Son :

```
1: src ← address(S.d[0])
2: dst ← address(T.d[x])
3: i ← 0
4: while i ⩽ 3G − x do
5:     t ← [00000000, ..., 00000000]            ▷ 8 bytes equal to 00000000
6:     pcmpeqb(src,t)
7:     pandn([00000001,...,00000001],t)
8:     psub(dst,t)
9:     dst ← t              ▷ copy the array [t[0], ..., t[7]] to [d[0], ..., d[7]]
10:    increase src,dst,i by 8
11: end while
```

Let us explain how this works in more details. Let i be an integer in $\{0, ..., 7\}$. After Line `5`, the value of `t`$[i]$ is `0000000`. After Line `6`, we have

$$\texttt{t}[i] = \begin{cases} \texttt{11111111} & \text{if } \texttt{src}[i] = \texttt{00000000} \text{ (corresponding to 0)} \\ \texttt{00000000} & \text{otherwise} \end{cases}$$

Line `7` performs a logical `and` between `00000001` and `not t`$[i]$. Hence the result byte is `00000001` if the last byte of `t`[i] is 0 and `00000000` otherwise:

$$\texttt{t}[i] = \begin{cases} \texttt{00000000} & \text{if } \texttt{src}[i] = \texttt{00000000} \text{ (corresponding to 0)} \\ \texttt{00000001} & \text{otherwise} \end{cases}$$

Line `8` subtracts `t`[i] (which is equal to 0 or 1) from `dst`$[i]$ according to Proposition 2.6. Finally we shift the cursor `src` and `dst` by eight cases.

Our second optimization is to use parallelism on exploration of the tree. Today, CPU of personal computer have several cores (2, 4 or more). The given version of our exploration algorithm uses a single core and so a fraction only of the power of a CPU.

Our method of parallelism is very simple: for $G \in \mathbb{N}$, we cut the tree $\mathcal{T}^{\leqslant G}$ in sub-trees $\mathcal{T}_1, ..., \mathcal{T}_n$ and we launch our algorithm on these sub-trees. The advantage of this method is that there is no communication between the instances of our algorithm. The disadvantage is that the cutting controls the efficiency of the parallelism. Assume for example that we want to explore the tree $\mathcal{T}^{\leqslant 40}$. We first determine all numerical semigroups $S_1, ...S_{n_{20}}$ of genus 20. To explore $\mathcal{T}$ it remains to explore the tree $\mathcal{T}_i$ rooted in S_i for $i = 1, ..., n_{20}$. This works but the time to explore $\mathcal{T}_1$ is similar to the time need to explore the tree $\mathcal{T}^{\leqslant 40}$. And in this case, using many cores does not reduce the time to explore the tree.

Let us now explain in more detail how we cut the tree $\mathcal{T}^{\leqslant G}$ in order to use parallelism. Semigroups of the form $\langle g+1, ..., 2g+1\rangle$, which are of genus g, are called *ordinary* in [8]. Each ordinary semigroup has a unique son that is also ordinary. We define X to be the set of all the non-ordinary sons of an ordinary semigroup and $X^{\leqslant G}$ the restriction of X to semigroup of genus $\leqslant G$. We then denote by $\mathcal{T}_i$ the tree rooted on S_i where $X^{\leqslant G} = \{S_1, ..., S_n\}$. The time needed to explore $\mathcal{T}_i$ is very heterogeneous but there are many tree T_i with maximal time. This cutting is more efficient than the previous one.

Figure 5 summarizes the time complexity of various exploration algorithms.

The version $depth - expl - \delta - mmx$ of our algorithm compute the value of n_g for $g \leqslant 50$ in 196 minutes on the `i5-3570K` CPU while the parallel version running on the 4 cores of the same CPU end the work in 50 minutes.

Using two `i7` based computers and our parallel algorithm we computed in two days the values of n_g for $g \leqslant 60$, confirming the values given by M.Bras-Amorós and M.Delgado :

g	n_g	n_g/n_{g-1}
0	1	
1	1	1.0
2	2	2.0
3	4	2.0
4	7	1.75
5	12	1.71428
6	23	1.91666
7	39	1.69565
8	67	1.71794
9	118	1.76119
10	204	1.72881
11	343	1.68137
12	592	1.72594
13	1001	1.69087
14	1693	1.69130
15	2857	1.68753
16	4806	1.68218
17	8045	1.67394
18	13467	1.67395
19	22464	1.66807
20	37396	1.66470
21	62194	1.66311
22	103246	1.66006
23	170963	1.65588
24	282828	1.65432
25	467224	1.65197
26	770832	1.64981
27	1270267	1.64791
28	2091030	1.64613
29	3437839	1.64408

g	n_g	n_g/n_{g-1}
30	5646773	1.64253
31	9266788	1.64107
32	15195070	1.63973
33	24896206	1.63843
34	40761087	1.63724
35	66687201	1.63605
36	109032500	1.63498
37	178158289	1.63399
38	290939807	1.63304
39	474851445	1.63212
40	774614284	1.63127
41	1262992840	1.63047
42	2058356522	1.62974
43	3353191846	1.62906
44	5460401576	1.62841
45	8888486816	1.62780
46	14463633648	1.62723
47	23527845502	1.62668
48	38260496374	1.62617
49	62200036752	1.62569
50	101090300128	1.62524
51	164253200784	1.62481
52	266815155103	1.62441
53	433317458741	1.62403
54	703569992121	1.62368
55	1142140736859	1.62335
56	1853737832107	1.62303
57	3008140981820	1.62274
58	4880606790010	1.62246
59	7917344087695	1.62220
60	12841603251351	1.62195

In [9], A.Zhai establishes that the limit of the quotient $\frac{n_g}{n_{g-1}}$, when g go to $+\infty$, is the golden ratio $\phi \approx 1.618$. As the reader can see the convergence is very slow.

References

[1] N. Sloane, "The on-line encyclopedia of integer sequences."

[2] M. Bras-Amorós, "Fibonacci-like behavior of the number of numerical semigroups of a given genus," *Semigroup Forum*, vol. 76, no. 2, pp. 379–384, 2008.

[3] M. Delgado, "Homepage."

[4] J. C. Rosales, "Fundamental gaps of numerical semigroups generated by two elements," *Linear Algebra Appl.*, vol. 405, pp. 200–208, 2005.

[5] J. M. Manuel Delgado, Pedro A. Garcia-Sanchez, *NumericalSgps – GAP package, Version 0.971*. T, 2011.
[6] The GAP Group, *GAP – Groups, Algorithms, and Programming, Version 4.6.3*, 2013.
[7] J. C. Rosales and P. A. García-Sánchez, *Numerical semigroups*, vol. 20 of *Developments in Mathematics*. New York: Springer, 2009.
[8] S. Elizalde, "Improved bounds on the number of numerical semigroups of a given genus," *J. Pure Appl. Algebra*, vol. 214, no. 10, pp. 1862–1873, 2010.
[9] A. Zhai, "Fibonacci-like growth of numerical semigroups of a given genus," *Semigroup Forum*, vol. 86, no. 3, pp. 634–662, 2013.

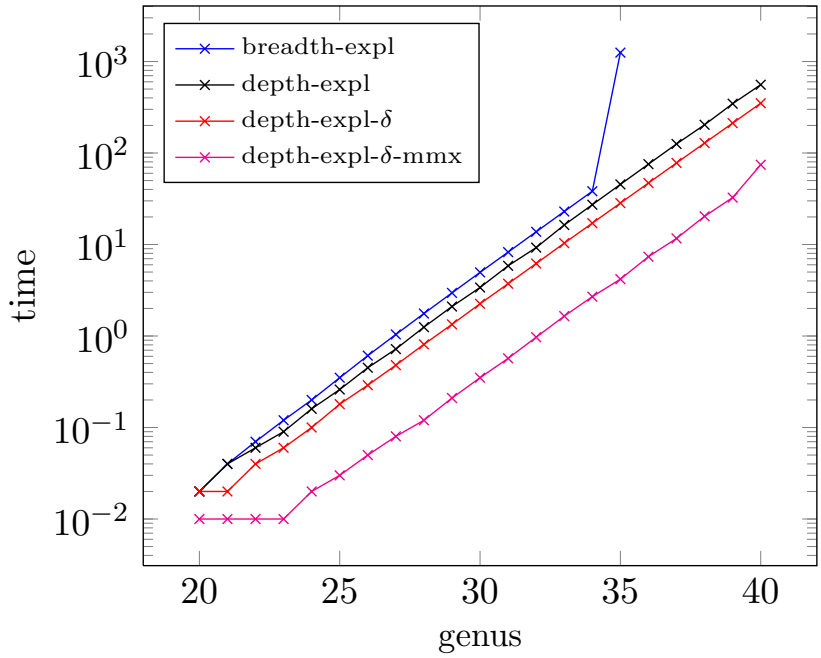

FIGURE 2. Comparison of time executions of different exploration algorithms of the tree $\mathcal{T}^{\leqslant g}$ on an Intel `i5-3570K` CPU with $8GB$ of memory. The scale is logarithmic in time. The first version is based on a breadth-first search exploration and the peak at genus 35 is due to the consumption of all the memory and use of swap. The second version uses a depth first search exploration. The third is based on the second one with use of decomposition numbers, it corresponds to the algorithm given in Section 4. The last one is an optimisation of the third one with MMX.