LMS J. Comput. Math. 0 (0) (2016) 1–20
doi:10.1112/S1461157016000061

Numerical semigroups with a given set of pseudo-Frobenius numbers

M. Delgado, P. A. García-Sánchez and A. M. Robles-Pérez

Abstract

The pseudo-Frobenius numbers of a numerical semigroup are those gaps of the numerical semigroup that are maximal for the partial order induced by the semigroup. We present a procedure to detect if a given set of integers is the set of pseudo-Frobenius numbers of a numerical semigroup and, if so, to compute the set of all numerical semigroups having this set as set of pseudo-Frobenius numbers.

1. *Introduction*

Let S be a numerical semigroup, that is, a cofinite submonoid of $(\mathbb{N}, +)$, where $\mathbb{N}$ stands for the set of non-negative integers.

An integer x is said to be the *Frobenius number* of S (respectively, a *pseudo-Frobenius number* of S) if $x \notin S$ and $x + s \in S$ for all $s \in \mathbb{N} \setminus \{0\}$ (respectively, for all $s \in S \setminus \{0\}$).

Given a positive integer f, there exist numerical semigroups whose Frobenius number is f. One example of such a semigroup is the semigroup $\{0, f + 1, \rightarrow\}$ containing 0 and all the integers greater than f. There are several algorithms to compute all the numerical semigroups with a given Frobenius number (the fastest we know is based on [**9**]).

We denote by $\mathrm{F}(S)$ the Frobenius number of S and by $\mathrm{PF}(S)$ the set of pseudo-Frobenius numbers of S. The cardinality of $\mathrm{PF}(S)$ is said to be the *type* of S and is denoted by $\mathrm{t}(S)$.

A positive integer that does not belong to S is said to be a *gap* of S and an element of S that is not greater than $\mathrm{F}(S) + 1$ is said to be a *small element* of S. To denote the set $\mathbb{N} \setminus S$ of gaps of S we use $\mathrm{gaps}(S)$ and to denote the set of small elements of S we use $\mathrm{smalls}(S)$. Since a set of gaps must contain the divisors of all its members and a set of small elements must contain all multiples of its members (up to its maximum), it is clear that there are sets of positive integers that cannot be the set of gaps or the set of small elements of a numerical semigroup. The set of gaps, as well as the set of small elements, completely determines the semigroup. Observe that when some elements or some gaps are known, others may be forced. For instance, a gap forces all its divisors to be gaps.

Let n be a positive integer and let $\mathsf{PF} = \{g_1, g_2, \ldots, g_{n-1}, g_n\}$ be a set of positive integers. Denote by $\mathcal{S}(\mathsf{PF})$ the set of numerical semigroups whose set of pseudo-Frobenius numbers is PF. When $n > 1$, $\mathcal{S}(\mathsf{PF})$ may clearly be empty. Moreover, when non-empty, it is finite.

Received 23 September 2015.

2010 Mathematics Subject Classification 11D07, 20M14, 20–04.

The first author was partially supported by CMUP (UID/MAT/00144/2013), which is funded by FCT (Portugal) with national (MEC) and European (FEDER) structural funds, under the partnership agreement PT2020. He also acknowledges the hospitality of the University of Granada during the various visits made, in particular one supported by the GENIL SSV program. The second and third authors are partially supported by the project FQM-343 (Junta de Andalucía). The second author is also supported by Junta de Andalucía/Feder grant no. FQM-5849. The third author is also supported by the Plan Propio (Plan de Fortalecimiento de los Grupos de Investigación) of the Universidad de Granada. The authors are also supported by the project MTM2014-55367-P, which is funded by Ministerio de Economía y Competitividad and Fondo Europeo de Desarrollo Regional FEDER.

In fact, $\mathcal{S}(\mathsf{PF})$ consists of semigroups whose Frobenius number is the maximum of PF. Some questions arise naturally. Among them, we can consider the following.

QUESTION 1. Find conditions on the set PF that ensure that $\mathcal{S}(\mathsf{PF}) \neq \emptyset$.

QUESTION 2. Find an algorithm to compute $\mathcal{S}(\mathsf{PF})$.

Both questions have been solved for the case that the set PF consists of a single element (which must be the Frobenius number of a numerical semigroup, symmetric numerical semigroups) or when PF consists on an even positive integer f and $f/2$ (pseudo-symmetric numerical semigroups), see [1]. Moreover, Question 1 was solved by Robles-Pérez and Rosales [7] in the case where PF consists of two elements (not necessarily of the form $\{f, f/2\}$).

The set $\mathcal{S}(\mathsf{PF})$ can be computed by choosing those semigroups that have PF as set of pseudo-Frobenius numbers from the set of numerical semigroups whose Frobenius number is the maximum of PF. Due, in part, to the possibly huge number of semigroups with a given Frobenius number, this is a rather slow procedure and we consider it far from being a satisfactory answer to Question 2 (there are 1 156 012 numerical semigroups with Frobenius number 39, and the set for Frobenius number 100 is by now out of reach).

Irreducible numerical semigroups with odd Frobenius number correspond with symmetric numerical semigroups, and those with even Frobenius number with pseudo-symmetric ones (see for instance [8, Chapter 3]). Bresinsky proved in [2] that symmetric numerical semigroups with embedding dimension 4 have minimal presentations of cardinality 5 or 3 (complete intersections). Symmetry of a numerical semigroup S translates to having $\{\mathrm{F}(S)\}$ as set of pseudo-Frobenius numbers. Later, Komeda [6] was able to prove the same result for pseudo-symmetric numerical semigroups (though he used different terminology for this property; in this setting 3 does not occur since pseudo-symmetric semigroups are never complete intersections). A numerical semigroup S is pseudo-symmetric if its set of pseudo-Frobenius numbers is $\{\mathrm{F}(S), \mathrm{F}(S)/2\}$. It should be interesting to see the relationship with the type and the cardinality of a minimal presentation, and thus having tools to find semigroups with given sets of pseudo-Frobenius numbers becomes helpful. Watanabe and his students Nari and Numata are making some progress in the study of this relationship.

1.1. *Contents*

We present two different procedures to determine the set of all numerical semigroups with a given set of pseudo-Frobenius numbers. One exploits the idea of irreducible numerical semigroups. From each irreducible numerical semigroup, we start removing minimal generators with certain properties to build a tree whose leaves are the semigroups we are looking for. The other approach is based on determining the elements and gaps of any numerical semigroup with the given set of pseudo-Frobenius numbers, obtaining in this way a list of 'free' integers. We then construct a binary tree in which branches correspond to assuming that these integers are either gaps or elements.

We start this work with some generalities and basic or well-known results. Then we describe a procedure to compute forced integers. Computing forced integers is fast and leads frequently to the conclusion that there exists no semigroup fulfilling the condition of having the given set as set of pseudo-Frobenius numbers. The following section of the paper is devoted to the above-mentioned approach of constructing a tree with nodes lists of integers, which turns out to be faster most of the time than the one based on irreducible numerical semigroups. Nevertheless, besides being useful to compare results, the method using irreducible numerical semigroups is of theoretical importance so we decided to keep it, and it is given in Appendix A. In Appendix B, we describe an algorithm that (increasing the number of attempts, if necessary)

returns one numerical semigroup with the given set of pseudo-Frobenius numbers, when such a semigroup exists. This procedure is reasonably fast in practice.

We give pseudo-code for various algorithms used in the approach of lists of free integers. For the implementation of the algorithms we have taken advantage of the existence of the GAP [5] package numericalSgps [4] for computing with numerical semigroups. However, all the algorithms use elementary operations and could be implemented in any language.

Many examples are given throughout the paper, some to illustrate the methods proposed, while others are included to motivate the options we have followed, and why we discarded other possible choices. In some of the examples we show the output obtained in a GAP session. These usually correspond to examples that are not suitable to a full computation just using a pencil and a sheet of paper; furthermore, indicative running times, as given by GAP, are shown (mainly in §§ 3 and 4).

A new version of the numericalSgps package, including implementations of the algorithms developed in the present work, is released at the same time as this work is made public. The implementations are available at the links given in the references (this software is open source). Hence, the interested reader can find a sharper depiction of the algorithms presented in this paper, and can use these implementations for testing examples (and produce new code based on them).

2. *Generalities and basic results*

Throughout the paper we will consider often a set $\mathsf{PF} = \{g_1, g_2, \ldots, g_{n-1}, g_n\}$ of positive integers. We will usually require it to be ordered, that is, we will assume that $g_1 < g_2 < \ldots < g_{n-1} < g_n$. For convenience, we write $\mathsf{PF} = \{g_1 < g_2 < \ldots < g_{n-1} < g_n\}$ in this case.

We denote by frob the maximum of PF and by type the cardinality of PF. Note that if $S \in \mathcal{S}(\mathsf{PF})$, then $\mathsf{frob} = g_n = \mathrm{F}(S)$ and $\mathsf{type} = n = \mathrm{t}(S)$.

2.1. *Well-known results*

The partial order $\leqslant_S$ induced by the numerical semigroup S on the integers is defined as follows: $x \leqslant_S y$ if $y - x \in S$. The following result is well known and will be used several times throughout this paper.

LEMMA 3 [8, Lemma 2.19]. *Let S be a numerical semigroup. Then:*
(i) $\mathrm{PF}(S) = \mathrm{Maximals}_{\leqslant_S}(\mathbb{Z} \setminus S)$;
(ii) *$x \in \mathbb{Z} \setminus S$ if and only if $f - x \in S$ for some $f \in \mathrm{PF}(S)$.*

The type of a numerical semigroup S is upper bounded by the least positive element belonging to S, which is known as the *multiplicity* of S and is denoted by $\mathrm{m}(S)$.

LEMMA 4 [8, Corollary 2.23]. *Let S be a numerical semigroup. Then* $\mathrm{m}(S) \geqslant \mathrm{t}(S) + 1$.

2.2. *Forced integers*

We say that an integer ℓ is a *gap forced by* PF or a PF-*forced gap* if ℓ is a gap of all the numerical semigroups having PF as set of pseudo-Frobenius numbers. In particular, if there is no semigroup with PF as set of pseudo-Frobenius numbers, then every non-negative integer is a gap forced by PF. We use the notation $\mathcal{GF}(\mathsf{PF})$ to denote the set of PF-forced gaps. In symbols: $\mathcal{GF}(\mathsf{PF}) = \bigcap_{S \in \mathcal{S}(\mathsf{PF})} \mathrm{gaps}(S)$.

In a similar way, we say that an integer ℓ is an *element forced by* PF or a PF-*forced element* if ℓ is an element of all semigroups in $\mathcal{S}(\mathsf{PF})$. We use the notation $\mathcal{EF}(\mathsf{PF})$ to denote the

set of small PF-forced elements. In symbols: $\mathcal{EF}(\mathsf{PF}) = \bigcap_{S\in\mathcal{S}(\mathsf{PF})} \text{smalls}(S)$. Note also that if $\mathcal{S}(\mathsf{PF}) = \emptyset$, then $\mathcal{EF}(\mathsf{PF}) = \mathbb{N}$.

The union of the PF-forced gaps and PF-forced elements is designated by PF-*forced integers*. The following is a simple, but crucial, observation.

PROPOSITION 5. *$\mathcal{S}(\mathsf{PF}) \neq \emptyset$ if and only if $\mathcal{GF}(\mathsf{PF}) \cap \mathcal{EF}(\mathsf{PF}) = \emptyset$.*

Proof. If $\mathcal{S}(\mathsf{PF}) = \emptyset$, then all non-negative integers are at the same time gaps and elements forced by PF. Conversely, assume that $\mathcal{S}(\mathsf{PF}) \neq \emptyset$ and let $S \in \mathcal{S}(\mathsf{PF})$. Then $\mathcal{GF}(\mathsf{PF}) \cap \mathcal{EF}(\mathsf{PF}) \subseteq \text{gaps}(S) \cap \text{smalls}(S) = \emptyset$. □

Frequently the prefix PF is understood and we will abbreviate by saying just *forced gap*, *forced element* or *forced integer*.

Let G and E be, respectively, sets of forced gaps and forced elements. The integers $v \in \{1, \ldots, \mathsf{frob}\}$ that do not belong to $G \cup E$ are said to be *free integers for* (G, E). When the pair (G, E) is understood, we simply call *free integer* a free integer for (G, E).

2.3. *Forced gaps*

The maximality of the pseudo-Frobenius numbers of S with respect to $\leqslant_S$ means that they are incomparable with respect to this ordering. In particular, the difference of any two distinct pseudo-Frobenius numbers does not belong to S, that is,

$$\{g_i - g_j \mid i, j \in \{1, \ldots, \mathrm{t}(S)\}, i > j\} \subseteq \text{gaps}(S). \tag{1}$$

This is the underlying idea of the next result.

LEMMA 6. *Let S be a numerical semigroup and suppose that $\mathrm{PF}(S) = \{g_1 < g_2 < \ldots < g_{n-1} < g_n\}$ with $n > 1$. Let $i \in \{2, \ldots, \mathrm{t}(S)\}$ and $g \in \langle \mathrm{PF}(S)\rangle$ with $g < g_i$. Then $g_i - g \in \text{gaps}(S)$.*

Proof. Assume that $g = g_{i_1} + \ldots + g_{i_k}$ for some $k \in \mathbb{N}$. We proceed by induction on k. The case $k = 1$ is given by (1). Assume that the result holds for $k - 1$ and let us prove it for k. If $g_i - g \in S$, then $g_{i_k} + (g_i - g) \in S$ by definition of pseudo-Frobenius number. It follows that $g_i - (g_{i_1} + \ldots + g_{i_{k-1}}) \in S$, contradicting the induction hypothesis. □

REMARK 7. Lemma 4 implies that $\{x \in \mathbb{N} \mid 1 \leqslant x \leqslant \mathrm{t}(S)\} \subseteq \text{gaps}(S)$. Hence, $\{1, \ldots, n\} \subseteq \mathcal{GF}(\mathsf{PF})$.

As the pseudo-Frobenius numbers of S are gaps of S and any positive divisor of a gap must be a gap also, we conclude that the set of divisors of

$$\mathrm{PF}(S) \cup \{x \in \mathbb{N} \mid 1 \leqslant x \leqslant \mathrm{t}(S)\} \cup \{g_i - g \mid i \in \{2, \ldots, \mathrm{t}(S)\}, g \in \langle \mathrm{PF}(S)\rangle, g_i > g\}$$

consists entirely of gaps of S.

Consider the set

$$\mathsf{PF} \cup \{x \in \mathbb{N} \mid 1 \leqslant x \leqslant n\} \cup \{g_i - g \mid i \in \{2, \ldots, n\}, g \in \langle \mathsf{PF}\rangle, g_i > g\}$$

and denote by $\text{sfg}(\mathsf{PF})$ the set of its divisors (as we are only considering positive divisors, in what follows we will not include this adjective). If S is a numerical semigroup such that $\mathrm{PF}(S) = \mathsf{PF}$, we deduce that $\text{sfg}(\mathsf{PF}) \subseteq \text{gaps}(S)$. We have proved the following result for the case where there is a numerical semigroup S such that $\mathrm{PF}(S) = \mathsf{PF}$. If no such semigroup exists, then $\mathcal{GF}(\mathsf{PF}) = \mathbb{N}$ and the result trivially holds.

COROLLARY 8. *Let* PF *be a set of positive integers. Then* $\mathrm{sfg}(\mathsf{PF}) \subseteq \mathcal{GF}(\mathsf{PF})$.

We use the terminology *starting forced gap for* PF to designate any element of $\mathrm{sfg}(\mathsf{PF})$, since $\mathrm{sfg}(\mathsf{PF})$ is the set we start with when we are looking for forced gaps.

REMARK 9. Whenever we find a new forced element, it might produce new forced gaps. The idea is the following. If e is an element and f is a gap, then $f - e$ is either negative or a gap. The divisors of these new gaps will also be forced gaps.

2.4. *Forced elements*

We use two ways to get new forced elements. One of these ways makes use of Lemma 3(ii). We refer to the elements obtained in this way as *elements forced by exclusion*.

LEMMA 10. *Assume that we have a set of forced gaps FG. Let $x \in FG$. Consider the set $H = \{h \in \mathsf{PF} \mid h - x \geqslant 0 \text{ and } h - x \notin FG\}$. If $H = \{h\}$ for some positive integer h, then $h - x$ is a forced element.*

Proof. Let $S \in \mathcal{S}(\mathsf{PF})$. As $x \in FG$, $x \notin S$ and then by Lemma 3(ii), there exists $g \in \mathsf{PF}$ such that $g - x = s \in S$. Hence, $g \in H$ and consequently $g = h$. This implies that $g - x = h - x = s \in S$. □

LEMMA 11. *Let FG be a set of forced gaps with $\mathsf{PF} \subseteq FG$. Take $x \in \{1, \ldots, \mathsf{frob} - 1\} \setminus FG$. If there is no positive integer in $(-x + \mathsf{PF}) \setminus FG$, then x is a forced element.*

Proof. Let S be a numerical semigroup with set of pseudo-Frobenius numbers PF. Assume that $x \notin S$. Then, according to Lemma 3(ii), there exist $f \in \mathsf{PF}$ and $s \in S$ such that $f = x + s$. Observe that $s \neq 0$, because $x \notin FG$. But then $s = -x + f \in (-x + \mathsf{PF}) \setminus FG$, which is a contradiction. □

Another way to find more forced elements makes use of the following lemma, which tells us that small gaps force elements that are close to the maximum of PF. Sometimes we refer to them by using the more suggestive terminology *big forced elements*.

LEMMA 12. *Let m be the multiplicity of a numerical semigroup S and let i be an integer such that $1 \leqslant i < m$. Then either $\mathrm{F}(S) - i \in S$ or $\mathrm{F}(S) - i \in \mathrm{PF}(S)$.*

Proof. It suffices to observe that, as $i < m$, one has that $\mathrm{F}(S) - i + s > \mathrm{F}(S)$ for all $s \in S \setminus \{0\}$. Consequently, $\mathrm{F}(S) - i + s \in S$ for all $s \in S \setminus \{0\}$. The result follows immediately from the definition of pseudo-Frobenius numbers. □

Since we do not know the multiplicity of the semigroups we are looking for, in the above result we can take m to be the least positive integer that is not in the current set of forced gaps.

REMARK 13. If fe is a list of forced elements, then so is $\langle \mathsf{fe} \rangle \cap \{0, \ldots, \mathsf{frob} - 1\}$. So, we will always take this closure when computing forced elements.

2.5. *Stop conditions*

We present here some obvious conditions on the set PF that can be used to stop the algorithms if they are not fulfilled, producing in this case the empty list.

Let $n > 1$ be an integer and let $\mathsf{PF} = \{g_1 < \ldots < g_n\}$ be a set of positive integers.

LEMMA 14. *Let S be a numerical semigroup such that $\mathrm{PF}(S) = \mathsf{PF}$. Let $i \in \{2, \ldots, n\}$ and $g \in \langle \mathsf{PF} \rangle \setminus \{0\}$ with $g < g_i$. Then there exists $k \in \{1, \ldots, n\}$ such that $g_k - (g_i - g) \in S$.*

Proof. Lemma 6 ensures that $g_i - g \notin S$. The conclusion follows from Lemma 3. □

By choosing $i = n$ in the above result, there exists $k \neq n$ such that $g_k - (g_n - g_1) \geqslant 0$ and $g_k - (g_n - g_1) \notin \{1, \ldots, \mathsf{type}\}$. (Note that $k = n$ would imply $g_1 \in S$, which is impossible.) But then $g_{n-1} - (g_n - g_1) \geqslant 0$, since $g_{n-1} \geqslant g_k$ for all $k \in \{1, \ldots, n-1\}$. We have thus proved the following corollary.

COROLLARY 15. *Let S be a numerical semigroup such that $\mathrm{PF}(S) = \mathsf{PF}$. Then $g_1 \geqslant g_n - g_{n-1}$.*

The computational cost of testing the condition $g_1 \geqslant g_n - g_{n-1}$ obtained in Corollary 15 is negligible and should be made before calling any procedure to compute $\mathcal{S}(\mathsf{PF})$, avoiding in many cases an extra effort that would lead to the empty set.

Other conditions of low computational cost would also be useful. Since $g_{n-1} - (g_n - g_1) \geqslant 0$, one could be tempted to ask whether replacing g_1 by g_2 one must have $g_{n-1} - (g_n - g_2) \notin \{1, \ldots, \mathsf{type}\}$ (since $\{1, \ldots, \mathsf{type}\}$ consists of gaps). The following example serves to rule out this one that seems to be a natural attempt.

EXAMPLE 16. Let $S = \langle 8, 9, 10, 11, 13, 14, 15 \rangle$. One can check easily that $\mathrm{PF}(S)$ equals $\{5, 6, 7, 12\}$. For $\mathsf{PF} = \{5, 6, 7, 12\}$, we have $g_{n-1} - (g_n - g_2) = 7 - (12 - 6) = 1 \in \{1, \ldots, 4\} = \{1, \ldots, \mathsf{type}\}$.

REMARK 17. In light of Lemma 3, we have another condition. For every element x in $\mathrm{sfg}(\mathsf{PF}) \setminus \mathsf{PF}$, there is always a positive integer in $(-x + \mathsf{PF}) \setminus \mathrm{sfg}(\mathsf{PF})$.

REMARK 18. It is also clear that at any step of the execution of the procedure, the set of forced gaps cannot intersect the set of forced elements. If this is not the case, the output must be empty (see Proposition 5).

Let us see how this last remark is used in an example.

EXAMPLE 19. Let $PF = \{4, 9\}$. Taking divisors and the difference $9 - 4$, one immediately sees that the set of starting forced gaps contains $\{1, 2, 3, 4, 5, 9\}$. But then 5 appears as a forced gap and as a (big) forced element. This is a contradiction, which shows that $\{4, 9\}$ cannot be the set of pseudo-Frobenius numbers of a numerical semigroup.

EXAMPLE 20. Figure 1 has been produced by using the `intpic` package [3]. It represents the set of numerical semigroups having $\{19, 29\}$ as set of pseudo-Frobenius numbers, and it is meant to illustrate the sets described in this section.

For $\mathsf{PF} = \{19, 29\}$, we have that the set $\{1, 2, 4, 5, 10, 11, 19, 20, 29\}$ consists of forced gaps and that the set $\{0, 9, 18, 24, 25, 27, 28, 30\}$ consists of forced elements. To the elements in each of these sets, as well as the ones in the sets of minimal generators, is assigned one color (*red* corresponds to pseudo-Frobenius numbers, *blue* to minimal generators, *green* to elements, *cyan* to forced gaps and *magenta* to forced elements; in a black and white visualization of this paper this will correspond with different gray tonalities). For integers that belong to more that one set, gradient colors are assigned.

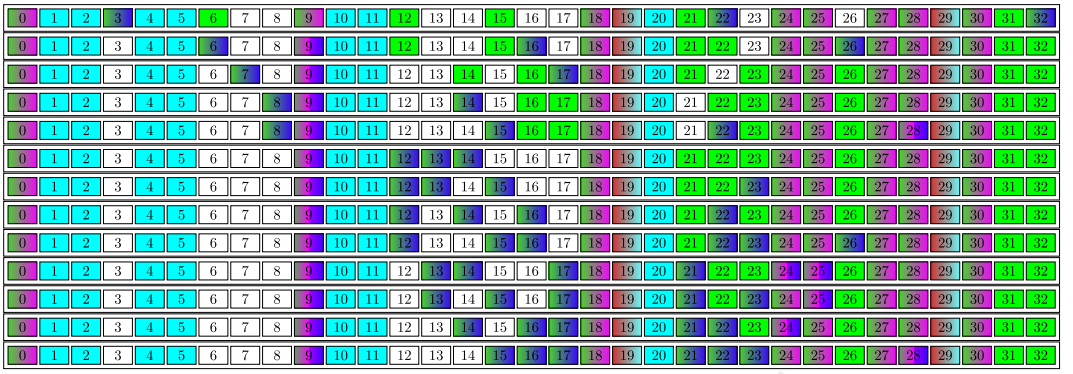

FIGURE 1. *The numerical semigroups with pseudo-Frobenius numbers* $\{19, 29\}$.

This picture as well as many others that we produced were fundamental for the design of the general procedure we describe in this paper.

3. *Computing forced integers*

In this section we present procedures (Algorithms 1 and 2) to compute forced integers. The first one is the faster. Its input consists of sets of forced gaps and forced elements previously computed and is appropriate to be used within a recursive function. For the second one we introduce a subtlety (the concept of admissibility of integers) that sometimes leads to finding extra forced integers. This section ends with some examples and execution times comparing both approaches.

3.1. *A quick procedure to compute forced integers*

With the contents of § 2, it is not hard to show that Algorithm 1 computes a list containing as first component a set of forced gaps and as second component a set of forced elements.

Input : g, e, where g, e are sets of PF-forced gaps and PF-forced elements, respectively.
Output: $[\mathsf{fg}, \mathsf{fe}]$, where $\mathsf{fg} \supseteq \mathsf{g}$ is a set of PF-forced gaps and $\mathsf{fe} \supseteq \mathsf{e}$ is a set of PF-forced elements, or fail, when some inconsistency is discovered

repeat
  Compute new gaps using § 2.3 and store them in fg;
  Compute new elements using § 2.4 and store them in fe;
  if *some inconsistency arises (§ 2.5)* **then**
    return fail;
until *no new gaps or elements arise*;
return $[\mathsf{fg}, \mathsf{fe}]$;

Algorithm 1: SimpleForcedIntegers.

THEOREM 21. *Algorithm 1 correctly produces the claimed output.*

REMARK 22. In light of Remark 13, the second component of the list returned by Algorithm 1 is the set of small elements of a numerical semigroup.

EXAMPLE 23. Let us see how `SimpleForcedIntegers` works for $\mathsf{PF} = \{16, 29\}$. Contrary to what happens in Example 20, here every integer below frob is forced. Let us call this function with `g` as the output as an initial call to § 2.3 and `e` the empty set. Then $\mathtt{g} = \{1, 2, 4, 8, 13, 16, 29\}$.

The first call to the contents of § 2.3 yields no new integers, while the first call to § 2.4 yields

$$\{0, 3, 6, 9, 12, 15, 18, 21, 24, 25, 27, 28, 30\}$$

as current forced elements. Observe that 3 is forced by exclusion (note that $3 = 16 - 13$; also, $29 - 13 = 16$ and 16 is a forced gap); 25 is also forced by exclusion (note that $16 - 25 < 0$ and $29 - 25 = 4$ is a forced gap). Also, 21 is forced by exclusion, but for now we do not need to worry with the multiples of 3, because these will appear when taking the closure.

The second call to § 2.3 produces

$$\{1, 2, 4, 5, 7, 8, 10, 11, 13, 14, 16, 17, 20, 23, 26, 29\}$$

as current forced gaps. To check it, observe that $29 - 3 = 26$, $29 - 6 = 23$, $29 - 9 = 20$ and so on are forced gaps. But, then, 10 and 5 are (forced to be) gaps.

In the second call to § 2.4, we obtain 19 and 22 as forced integers.

The union of the sets of forced gaps and of forced elements is $\{0, \ldots, 30\}$. Therefore, all positive integers less than 29 are forced. One can check that the closure of the set of forced elements does not produce new forced elements and thus it is the set of small elements of a numerical semigroup. Also, one can check that no forced gap outside $\{16, 29\}$ is a pseudo-Frobenius number and thus one may conclude that there exists exactly one numerical semigroup S such that $\mathrm{PF}(S) = \{16, 29\}$.

This example illustrates that more than one passage through the *repeat-until* loop of the algorithm `SimpleForcedIntegers` may be needed.

EXAMPLE 24. Let us now apply the algorithm to $\mathsf{PF} = \{15, 20, 27, 35\}$. We will use GAP to help us in doing the calculations (which can be easily confirmed by hand).

```
gap> pf := [ 15, 20, 27, 35 ];;
gap> sfg := StartingForcedGapsForPseudoFrobenius(pf);
[ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 15, 20, 27, 35 ]
```

We immediately get that $\{25, 26, 28, 29, 30, 31, 32, 33, 34\}$ consists of forced big elements. And, one can observe that 19 and 23 are forced by exclusion. This leads to the obtention of $35 - 19 = 16$ as forced gap. No other forced elements are obtained, which agrees with the following continuation of the GAP session:

```
gap> SimpleForcedIntegersForPseudoFrobenius(sfg,[],pf);
[ [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 15, 16, 20, 27, 35 ],
  [ 0, 19, 23, 25, 26, 28, 29, 30, 31, 32, 33, 34, 36 ] ]
```

3.2. *A not so quick procedure to compute forced integers*

We start by introducing a concept that may lead to the discovering of new forced gaps. Algorithm 2 will make use of it. Finding further forced integers has a non-negligible cost in terms of time and using it in all the situations may not be the best option.

Let G and E respectively be sets of PF-forced gaps and PF-forced elements, and let v be a free integer for (G, E). We say that v is *admissible for* (G, E) if Algorithm 1 does not return *fail* when applied to $(G, E \cup \{v\})$. Otherwise, we say that v is *non-admissible for* (G, E). If v is non-admissible, then v cannot be an element of any semigroup in $\mathcal{S}(\mathsf{PF})$. Therefore, v is a PF-forced gap. We state this fact as a lemma.

LEMMA 25. *Let G and E be sets of forced gaps and forced elements, respectively. Let v be free for (G, E). If v is non-admissible for (G, E), then v is a* PF*-forced gap.*

The following example shows that there exist forced gaps that are not detected by Algorithm 1, but arise when looking for non-admissible integers.

EXAMPLE 26. Let $PF = \{11, 22, 23, 25\}$.

```
gap> pf := [ 11, 22, 23, 25 ];;
gap> sfg := StartingForcedGapsForPseudoFrobenius(pf);
[ 1, 2, 3, 4, 5, 6, 7, 11, 12, 14, 22, 23, 25 ]
```

By using the function `SimpleForcedIntegers`, one obtains the following.

```
gap> SimpleForcedIntegersForPseudoFrobenius(sfg,[],pf);
[ [ 1, 2, 3, 4, 5, 6, 7, 11, 12, 14, 22, 23, 25 ],
  [ 0, 18, 19, 20, 21, 24, 26 ] ]
```

One can easily confirm by hand that the set $\{1, 2, 3, 4, 5, 6, 7, 11, 12, 14, 22, 23, 25\}$ consists of forced gaps and that the set $\{0, 18, 19, 20, 21, 24, 26\}$ consists of forced elements.

Let us now check that 15 is non-admissible. If it was an element of a semigroup $S \in \mathcal{S}(PF)$, then $10\,(= 25 - 15)$ and $8\,(= 23 - 15)$ would be gaps of S. But then 17 is a big element, $13\,(= 23 - 10)$ is forced by exclusion (note that $25 - 10 = 15$, $22 - 10 = 12$ and $11 - 10 = 1$ are gaps) and $9\,(= 23 - 14)$ is forced by exclusion too ($25 - 14 = 11$, $22 - 14 = 8$ are gaps and $11 - 14 < 0$). This is not possible, since $13 + 9 = 22$ is a gap. Therefore, 15 is non-admissible.

The following remark may be used to reduce the number of calls to Algorithm 1 and thus improve the performance of a function to compute non-admissible integers.

REMARK 27. A semigroup generated by admissible elements for some pair (G, E) consists of admissible elements for (G, E).

Algorithm 2 is our procedure to compute forced integers that produces the best result in terms of the number of forced integers encountered. Apart from Algorithm 1, it makes use of the concept of non-admissible integers to find new forced gaps (which may then force others elements and gaps).

REMARK 28. It is a consequence of Remark 22 that the second component of the list returned by Algorithm 2 is the set of small elements of a numerical semigroup.

The correctness of Algorithm 2 follows from the discussion at the beginning of this subsection (including Lemma 25) and Theorem 21.

THEOREM 29. *Algorithm 2 correctly produces the claimed output.*

3.3. *Examples and execution times*

Algorithm 1 can be used as a quick way to compute forced integers. In fact, when called with the starting forced gaps and the empty set of elements, it can be seen as a quick version of Algorithm 2. Table 1 collects some information concerning some execution times (as given by GAP) and the numbers of forced gaps and of forced elements both using Algorithm 1 (identified as QV (which stands for *quick version*)) and Algorithm 2 (identified as NV (which stands for *normal version*)). We observe that the execution times when using the quick version remain relatively small, even when the Frobenius number is large.

Input : PF with $\mathsf{PF} \neq \{\mathsf{frob}\}$ and $\mathsf{PF} \neq \{\mathsf{frob}/2, \mathsf{frob}\}$
Output: fail if some inconsistency is discovered; otherwise, returns [fg, fe], where fg and fe are sets of forced gaps and forced elements, respectively.

```
fints := SimpleForcedIntegers (sfg(PF), ∅);
if fints = fail then
    return fail
else
    if fints[1] ∪ fints[2] = {0, ..., frob} then
        return fints;
newgaps := set of non-admissible integers for (fints[1],fints[2]);
return SimpleForcedIntegers (newgaps ∪ fints[1],fints[2]);
```

Algorithm 2: ForcedIntegers.

Failure is usually detected very quickly, as Table 2 somehow confirms.
As one could expect, there are examples where failure is not detected with the quick version.

TABLE 1. *Execution times for computing forced integers.*

Pseudo-Frobenius numbers	Time		#F. Gaps		#F. Els	
	QV	NV	QV	NV	QV	NV
[11, 22, 23, 25]	2	7	13	14	7	7
[17, 27, 28, 29]	2	5	16	17	8	10
[17, 19, 21, 25, 27]	2	9	15	16	8	8
[15, 20, 27, 35]	2	10	16	16	13	13
[12, 24, 25, 26, 28, 29]	3	11	18	22	6	9
[145, 154, 205, 322, 376, 380]	47	1336	82	85	52	54
[245, 281, 282, 292, 334, 373, 393, 424, 432, 454, 467]	129	2075	116	116	53	53
[223, 434, 476, 513, 549, 728, 828, 838, 849, 953]	213	5866	300	318	221	253
[219, 437, 600, 638, 683, 779, 801, 819, 880]	205	4838	219	224	153	161
[103, 110, 112, 137, 160, 178, 185]	25	262	50	51	29	31

TABLE 2. *When failure occurs ...*

Pseudo-Frobenius numbers	Time	
	QV	NV
[18, 42, 58, 88, 94]	6	6
[20, 27, 34, 35, 37, 42, 48, 80]	2	3
[30, 104, 118, 147, 197, 292, 298, 315, 333, 384, 408]	32	43
[36, 37, 219, 233, 304, 410, 413, 431, 438, 458]	35	32
[89, 411, 446, 502, 557, 600, 605, 631, 636, 796, 801, 915]	223	233
[56, 134, 136, 137, 158, 248, 277, 373, 383, 389, 487, 558, 566, 621, 691, 825, 836]	103	113

EXAMPLE 30. Let $\mathsf{PF} = \{25, 29, 33, 35, 38, 41, 46\}$.

```
gap> pf := [ 25, 29, 33, 35, 38, 41, 46 ];;
gap> ForcedIntegersForPseudoFrobenius(pf);
fail
gap> ForcedIntegersForPseudoFrobenius_QV(pf);
[ [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 16, 17, 19, 21,
  23, 25, 29, 33, 35, 38, 41, 46 ], [ 0, 30, 34, 36, 37, 39, 40,
  42, 43, 44, 45, 47 ] ]
```

We have not been able to detect any set PF candidate to be the set of pseudo-Frobenius numbers of a numerical semigroup that passes the normal version and such that $\mathcal{S}(\mathsf{PF}) = \emptyset$. Despite the various millions of tests made (some with the algorithm explained in Appendix B), we do not feel comfortable on leaving it as a conjecture; we leave it as a question instead.

QUESTION 31. If Algorithm 2 with input PF does not return *fail*, then $\mathcal{S}(\mathsf{PF}) \neq \emptyset$?

We observe that in view of the execution times illustrated in Table 1, a positive answer to Question 31 would imply that Algorithm 2 constitutes a satisfactory answer to Question 1.

4. *An algorithm based on forced integers*

In this section we present our main algorithm (Algorithm 3), which computes $\mathcal{S}(\mathsf{PF})$. Its correctness is stated in Theorem 39, whose proof is built from almost all the results preceding it in the paper.

After considering some initial cases, the algorithm makes a call to RecursiveDepthFirstExploreTree, which is a recursive function used to construct a tree whose nodes are labeled by pairs (X, Y), where X is a list of forced gaps and Y is a list of forced elements. Thus, we implicitly have lists of free integers in each node: the complement of $X \cup Y$ in the set $U = \{1, \ldots, g_n\}$, where $\mathsf{PF} = \{g_1 < \ldots < g_n\}$. Nodes with an empty set of free integers are the leaves in our tree.

A node (X, Y) such that there exists a numerical semigroup $S \in \mathcal{S}(\mathsf{PF})$ for which $X \subseteq \mathrm{gaps}(S)$ and $Y \subseteq \mathrm{smalls}(S)$ is said to be PF-*feasible*, or simply *feasible*, if PF is understood. A node that is not feasible is called a *dead node*.

REMARK 32. The knowledge of some forced integers allows us to identify immediately some dead nodes: if (G, E) is such that G consists of forced gaps and E consists of forced elements, then any node (X, Y) such that $X \cap E \neq \emptyset$ or $Y \cap G \neq \emptyset$ is a dead node.

REMARK 33. Let (X, Y) be a leaf that is not a dead node. It follows from the construction (see Remarks 22 and 28) that there exists a numerical semigroup S fulfilling that $(\mathrm{gaps}(S), \mathrm{smalls}(S)) = (X, Y)$.

REMARK 34. Observe that if $\mathsf{PF} = \{g_1\}$ or $\mathsf{PF} = \{g_1/2 < g_1\}$, then the set of numerical semigroups with pseudo-Frobenius numbers PF corresponds with the set of irreducible numerical semigroups having Frobenius number g_1; see Appendix A. In this case we will use the fast procedure presented in [1].

4.1. *The recursive construction of a tree*

A naive idea is to start with a list of free integers (for some set of forced integers) and turn each one of these free integers into either a gap or an element. Assuming that the number of

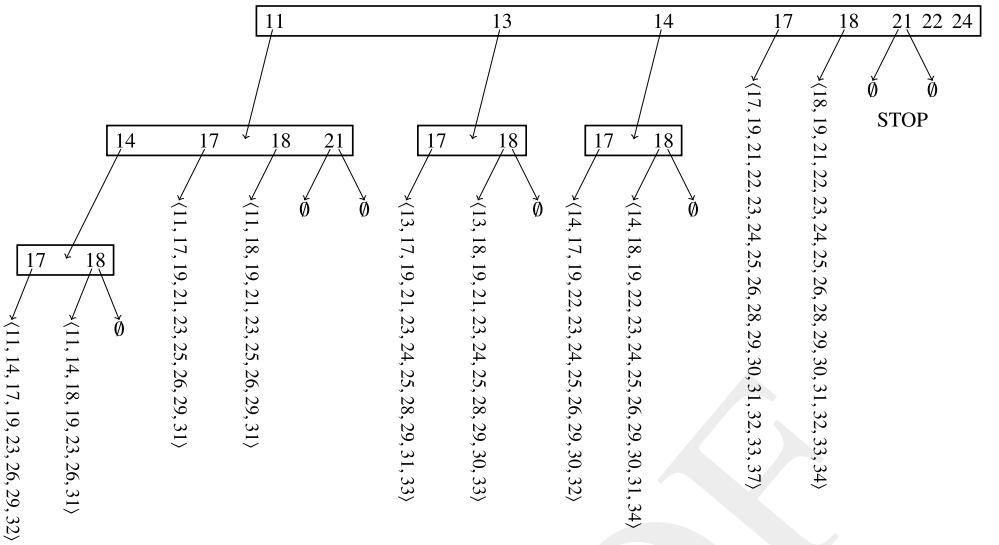

FIGURE 2. *The numerical semigroups with pseudo-Frobenius numbers* $\{15, 20, 27, 35\}$.

free integers is n, the number of possibilities is 2^n; thus, checking each of these possibilities for being in correspondence with the set of gaps of a numerical semigroup with PF as set of pseudo-Frobenius numbers is unfeasible, unless n is small. Nevertheless, this idea can evolve by taking advantage of the already observed facts that elements can force gaps and *vice versa*. Although there are examples for which fixing one integer does not force anything else, which lets us expect that nothing good can result from a worst-case complexity analysis, in practice it works quite well. We give some examples, but leave a detailed analysis of the complexity (perhaps average complexity) as an open problem.

The procedure we use is, for each integer v in the current list of free integers, compute all numerical semigroups containing v and the forced elements, and having the forced gaps as gaps. We proceed recursively and use backtracking when a semigroup or a dead node is found. When we revisit the node, we then suppose that v is a gap and continue with the next free integer.

Before proceeding with the pseudo-code for the recursive function RecursiveDepthFirstExploreTree that constructs the tree in a depth-first manner, let us give an example which we accompany by a picture (Figure 2).

EXAMPLE 35. Let $\mathsf{PF} = \{15, 20, 27, 35\}$. From Example 24, we have a pair of lists of forced gaps and forced integers, which leaves the list

$$F = \{11, 13, 14, 17, 18, 21, 22, 24\} \tag{2}$$

of free integers.

The leaves contained in the branch that descends from 11 in Figure 2 consist of the semigroups containing 11 as forced element.

All the remaining semigroups in $\mathcal{S}(\mathsf{PF})$ must have 11 as gap. The process then continues as Figure 2 illustrates: considering 13 as a forced integer, developing its corresponding subtree and so on.

Let us now look at the integer 21 in the root of the tree. At this point, all the semigroups in $\mathcal{S}(\mathsf{PF})$ containing some integer preceding 21 in F have been computed. Thus, any new semigroup must have the elements in $\{11, 13, 14, 17, 18\}$ as gaps. From the computations, one concludes that no new semigroup appears having 21 as an element. Thus, if some not already

computed semigroup exists fulfilling the current conditions, then it must have 21 as gap. One can check that this cannot happen. The remaining elements (22 and 24) need not be checked, since 21 had to be either an element or a gap. Therefore, we can stop.

Next we give pseudo-code for RecursiveDepthFirstExploreTree. The recursion ends when no more than one free integer is left. A call to the function EndingCondition is then made.

The variable **semigroups** works as a container which serves to store all the semigroups we are looking for, as they are discovered.

```
EndingCondition(g,e)
                                  ▷ g and e are such that #{1,...,frob} \ (g ∪ e) ⩽ 1
                        ▷ g and e represent lists of gaps and elements, respectively
free :={1,...,frob} \ (g ∪ e);
1  if Length(free) = 0 then
2      if there is no f ∈ PF \ g such that (f + e \ {0}) ∩ g is empty then
           Add to semigroups the numerical semigroup
           NumericalSemigroupByGaps(g));
       return;
3  if Length(free) = 1 then
       if g represents the set of gaps of a numerical semigroup S then
4          if there is no f ∈ PF \ g such that (f + (e ∪ free) \ {0}) ∩ g is empty then
               Add to semigroups the numerical semigroup S;
       if g ∪ free represents the set of gaps of a numerical semigroup S then
5          if there is no f ∈ PF \ g such that (f + e \ {0}) ∩ (g ∪ free) is empty then
               Add to semigroups the numerical semigroup S;
       return;
```

Function EndingCondition.

LEMMA 36. *Function EndingCondition either does nothing or adds to* ***semigroups*** *a numerical semigroup S such that* $\mathrm{PF}(S) = \mathsf{PF}$.

Proof. It suffices to observe that any of the conditions in the 'if' statements of Lines 2, 4 and 5 guarantee that no forced gap outside PF can be a pseudo-Frobenius number. □

Notice that when EndingCondition does nothing, it is because one of the following reasons:

- there is no numerical semigroup whose set of gaps is the first component of the input;
- the resulting semigroup does not have PF as set of pseudo-Frobenius numbers (it actually has more pseudo-Frobenius numbers);
- there is a free element that cannot be neither a gap nor an element.

Recall that recursion in RecursiveDepthFirstExploreTree ends (in Line 3) when there is at most one free integer.

PROPOSITION 37. *Let (fg, fe) be a pair of disjoint sets of integers contained in $U = \{0, \ldots, g_n+1\}$. After the execution of the function RecursiveDepthFirstExploreTree with input (fg, fe),* ***semigroups*** *contains all numerical semigroups S such that $S \in \mathcal{S}(\mathsf{PF})$, $fg \subseteq \mathrm{gaps}(S)$ and $fe \subseteq \mathrm{smalls}(S)$.*

```
RecursiveDepthFirstExploreTree([fg, fe])
                ▷ fg and fe consist of forced gaps and elements, respectively
currentfree := {1, ..., frob} \ (fg ∪ fe);
nfg := fg;                                        ▷ used to store new forced gaps
while Length(currentfree) > 1 do
    v := currentfree[1];
1   left := SimpleForcedIntegers (nfg,fe ∪ {v});
2   if left = fail then
        right := SimpleForcedIntegers (nfg ∪ {v}),fe);
        if (right = fail) or (right[1] ∩ right[2] ≠ ∅) then
            break;
    else
        RecursiveDepthFirstExploreTree ([left[1],left[2]]);
    nfg := nfg ∪ {v};
    currentfree := currentfree \ {v};
3 if Length(currentfree) ⩽ 1 then
    EndingCondition (fg,fe);
```

Function RecursiveDepthFirstExploreTree.

Proof. Denote by Λ the set of all numerical semigroups S such that $fg \subseteq \text{gaps}(S)$ and $fe \subseteq \text{smalls}(S)$. We have to prove that **semigroups** $\cap \Lambda = \mathcal{S}(\mathsf{PF}) \cap \Lambda$.

$\subseteq$. It suffices to observe that the numerical semigroups are added to **semigroups** by the function EndingCondition and these belong to $\mathcal{S}(\mathsf{PF})$, by Lemma 36.

$\supseteq$. Let $S \in \mathcal{S}(\mathsf{PF})$ be such that $fg \subseteq \text{gaps}(S)$ and $fe \subseteq \text{smalls}(S)$. If $\#U \setminus (fg \cup fe) \in \{0, 1\}$, then the function EndingCondition is called and it gives the correct output. Otherwise, we enter a recursion.

We will prove by induction on the number of free elements that the output is correct. Let us consider the smallest integer $v \in U \setminus (fg \cup fe)$. RecursiveDepthFirstExploreTree is called with input (fg, fe), and will enter the while loop, adding v to fe and computing new forced integers (`SimpleForcedIntegers`).

- If $v \in S$, then $left$ in Line 1 will not be equal to fail (Theorem 21), and we will call RecursiveDepthFirstExploreTree with a larger set fe, having in consequence fewer free integers. By induction, S is added to **semigroups**.
- Now assume that $v \notin S$. After the execution of the if–then–else starting in Line 2, v is added to the set of gaps. We have then one element less in the list of free integers, $fg \cup \{v\} \subseteq \text{gaps}(S)$ and $fe \subseteq \text{smalls}(S)$, whence the effect is the same as if we called RecursiveDepthFirstExploreTree with arguments $fg \cup \{v\} \subseteq \text{gaps}(S)$ and $fe \subseteq \text{smalls}(S)$. By the induction hypothesis, S is added to **semigroups**. □

Observe that if $\mathcal{S}(\mathsf{PF}) \neq \emptyset$ and the sets fg and fe considered in Proposition 37 consist of forced gaps and forced elements, respectively, then $\Lambda \subseteq \mathcal{S}(\mathsf{PF})$. Therefore, we have proved the following corollary.

COROLLARY 38. *If the function RecursiveDepthFirstExploreTree is called with a pair (fg, fe), where fg consists of forced gaps and fe consists of forced elements, then at the end of its execution we have that* **semigroups** $= \mathcal{S}(\mathsf{PF})$.

Input : PF
Output: the set $\mathcal{S}(\mathsf{PF})$.

if $\mathsf{PF} = \{\mathsf{frob}\}$ *or* $\mathsf{PF} = \{\mathsf{frob}/2, \mathsf{frob}\}$ **then**
  Use the procedure described in [1];
root:=`ForcedIntegers` (PF);
if root = fail **then**
  return $\emptyset$;
semigroups := $\emptyset$;
`RecursiveDepthFirstExploreTree` (root);
return semigroups;

Algorithm 3: NumericalSemigroupsWithPseudoFrobeniusNumbers.

4.2. *The main algorithm*

We observe that Algorithm 3 is not efficient when there are many free integers (for some set of forced integers). A possible attempt to improve it could be to replace the recursive function by a much more efficient depth-first search algorithm to parse the tree in question. In any case, we have not been able to write an iterative process to avoid recursiveness, which would be a better option from the point of view of programming.

Another possible approach is to use extra theoretical tools. What is done in Appendix A could be seen as an attempt. Having at our disposal more than one approach, we can take advantage of choosing the most efficient for each situation.

THEOREM 39. *Let* $\mathsf{PF} = \{g_1 < \ldots < g_n\}$ *be a set of positive integers. The output of Algorithm 3 with input* PF *is* $\mathcal{S}(\mathsf{PF})$.

Proof. As Algorithm 2 (`ForcedIntegers`) returns *fail* precisely when it has found some forced gap that is at the same time a forced element, Proposition 5 ensures us that $\mathcal{S}(\mathsf{PF}) = \emptyset$, which is the set returned by the algorithm.

When nothing of the above holds, the variable **semigroups** is initialized as the empty set and there is made a call to the recursive function RecursiveDepthFirstExploreTree. As we are considering this variable global to the functions EndingCondition, the result follows from Corollary 38. □

Algorithm 3 would also work with the initial call to `SimpleForcedIntegers` instead of `ForcedIntegers` (any inconsistency as in Example 30 will arise later) . The idea is to use the best filter in the starting steps of the algorithm, and leave the quick version for the recursion steps.

4.3. *Running times and examples*

The number of semigroups can be quite large compared to the number of free elements. The following example illustrate this possibility.

EXAMPLE 40.

```
gap>  pf := [ 68, 71, 163, 196 ];;
gap> forced := ForcedIntegersForPseudoFrobenius(pf);;
gap> free := Difference([1..Maximum(pf)],Union(forced));;
gap> Length(free);
```

```
38
gap> list := NumericalSemigroupsWithPseudoFrobeniusNumbers(pf);;
gap> Length(list);
1608
```

In the continuation of the previous GAP session, we do a kind of verification of the result.

```
gap> ns := RandomList(list);
<Numerical semigroup>
gap> MinimalGeneratingSystem(ns);
[ 35, 38, 65, 81, 89, 94, 99, 101, 104, 106, 109, 110, 112, 113,
  117, 118, 121, 122, 133 ]
gap> PseudoFrobeniusOfNumericalSemigroup(ns);
[ 68, 71, 163, 196 ]
```

Table 3 is meant to illustrate some timings, and the fact that there is no straight relationship between the number of numerical semigroups having a set of pseudo-Frobenius numbers and the number of free integers for this set of pseudo-Frobenius integers. Its content was produced by using repeatedly the commands of the first part of the previous example. The candidates to PF were obtained randomly. We observe that, although depending on some factors (such as the type or the Frobenius number we are interested in), the vast majority of the candidates would lead to the empty set. We do not consider them in this table (some were given in Table 2).

TABLE 3. *Some examples of execution data of the main algorithm.*

Pseudo-Frobenius numbers	#Free	#Semigroups	Time
[15, 27, 31, 43, 47]	0	1	3
[16, 30, 33, 37]	9	3	12
[40, 65, 80, 89, 107, 110, 130]	5	3	29
[32, 35, 44, 45, 48]	13	7	24
[40, 65, 89, 91, 100, 106]	24	9	99
[36, 50, 56, 57, 63]	25	39	123
[43, 50, 52, 65]	35	213	605
[68, 71, 163, 196]	38	1608	16 603
[38, 57, 67, 74, 79]	40	155	527
[68, 72, 76, 77]	46	177	607
[62, 78, 99, 129, 130]	53	4077	28 622
[128, 131, 146, 151, 180, 216, 224, 267, 271, 287]	54	954	24 253
[84, 103, 144, 202, 230, 242, 245]	56	14 292	277 094
[66, 85, 86, 92]	55	950	4683
[76, 79, 88, 102]	64	1409	6505
[61, 67, 94, 105]	69	4432	21 471
[114, 150, 179, 182, 231, 236, 254, 321]	69	302 929	7 121 020
[62, 73, 166, 190, 203]	77	9934	134 554
[102, 104, 118, 123, 134, 146, 149]	87	15 910	149 910

Appendix A. *An approach based on irreducible numerical semigroups*

We present here an alternative way to compute the set of numerical semigroups with a given set of pseudo-Frobenius numbers. In general, this procedure is slower than the one presented above, though we have not been able to characterize when this happens. We include it in the paper since it was the initial implementation and was used to test the other one.

A numerical semigroup is *irreducible* if it cannot be expressed as the intersection of two numerical semigroups properly containing it. It turns out that a numerical semigroup S is irreducible if and only if either $\mathrm{PF}(S) = \{\mathrm{F}(S)\}$ or $\mathrm{PF}(S) = \{\mathrm{F}(S)/2, \mathrm{F}(S)\}$ (see [8, Chapter 3]). Irreducible numerical semigroups can also be characterized as those maximal (with respect to set inclusion) numerical semigroups in the set of all numerical semigroups with given Frobenius number.

The maximality of irreducible numerical semigroups in the set of all numerical semigroups with given Frobenius number implies that every numerical semigroup is contained in an irreducible numerical semigroup with its same Frobenius number. Actually, we can say more.

LEMMA A.1. *Let S be a numerical semigroup. There exists an irreducible numerical semigroup T such that:*

(1) $\mathrm{F}(S) = \mathrm{F}(T)$;
(2) $]\mathrm{F}(S)/2, \mathrm{F}(S)[\cap \mathrm{PF}(S) \subset T$.

Proof. Let $f = \mathrm{F}(S)$ and let $F =]f/2, f[\cap \mathrm{PF}(S)$. We claim that $S' = S \cup F$ is a numerical semigroup with $\mathrm{F}(S') = f$. Take $s, s' \in S' \setminus \{0\}$.

- If both s and s' are in S, then $s + s' \in S \subseteq S'$.
- If $s \in S$ and $s' \in F$, then $s + s' \in S$, because $s' \in \mathrm{PF}(S)$.
- If $s, s' \in F$, then $s + s' > f$, and so $s + s' \in S \subseteq S'$.

Let us show that $\mathrm{F}(S') = f$. Assume to the contrary that this is not the case, and consequently $f \in S'$. Then, as all the elements in F are greater than $f/2$, and $f \notin S$, there must be elements $s \in S$ and $g \in \mathrm{PF}(S)$ such that $g + s = f$. But this is impossible, since all elements in $\mathrm{PF}(S)$ are incomparable with respect to $\leqslant_S$ (Lemma 3).

If S' is not irreducible, then, as irreducible numerical semigroups are maximal in the set of numerical semigroups with fixed Frobenius number, there exists T with $\mathrm{F}(S) = \mathrm{F}(S') = \mathrm{F}(T)$ and containing S'; whence fulfilling the desired conditions. □

With all these ingredients, we get the following result.

PROPOSITION A.2. *Let S be a non-irreducible numerical semigroup with $\mathrm{PF}(S) = \{g_1 < \ldots < g_k\}$, $k \geqslant 2$. Then there exists a chain*

$$S = S_0 \subset S_1 = S \cup \{x_1\} \subset \ldots \subset S_l = S \cup \{x_1, \ldots, x_l\}$$

with:

(1) *S_l irreducible;*
(2) *$x_i = \max(S_l \setminus S_{i-1})$ for all $i \in \{1, \ldots, l\}$;*
(3) *$]g_k/2, g_k[\cap \mathrm{PF}(S) \subset S_l$;*
(4) *for every $i \in \{1, \ldots, l\}$, x_i is a minimal generator of S_i such that $g_j - x_i \in S_i$ for some $j \in \{1, \ldots, k\}$;*
(5) *for every $i \in \{1, \ldots, l\}$ and $f \in \mathrm{PF}(S_i)$ with $f \neq g_k$, there exists $j \in \{1, \ldots, k-1\}$ such that $g_j - f \in S_i$.*

Proof. Let T be as in Lemma A.1. Construct a chain joining S and T by setting $S_0 = S$ and $S_i = S_{i-1} \cup \{x_i\}$ with $x_i = \max(T \setminus S_{i-1})$. Then S_i is a numerical semigroup, and x_i is a

minimal generator of S_i and a pseudo-Frobenius number for S_{i-1} [8, Lemma 4.35]. Since the complement $T \setminus S$ is finite, for some k, $S_k = T$.

Clearly, $x_1 = g_k$ and thus $x_1 - g_k = 0 \in S$. Now let $i \in \{2, \dots, k\}$. Then $x_i \in T \setminus S$ and, by Lemma 3, there exists $j \in \{1, \dots, k\}$ such that $g_j - x_i \in S \subseteq S_i$.

Take $f \in \mathrm{PF}(S_i) \setminus \{g_k\}$. Then $f \notin S$ and thus $g_j - f \in S$ for some $j \in \{1, \dots, k\}$ (Lemma 3 once more). Consequently, $g_j - f \in S_i$. Notice that $j \neq k$, since $f - g_k < 0$. □

Given a candidate set PF of pseudo-Frobenius numbers with maximum element f, we can use the above procedure to construct from the set of all irreducible numerical semigroups with Frobenius number f the set of all numerical semigroups having PF as a set of its pseudo-Frobenius numbers. In order to compute the set of all irreducible numerical semigroups with Frobenius number f, we use implementation of the procedure presented in [1] that is already part of [4]. We have slightly modified the algorithm in [1] to compute the set of irreducible numerical semigroups containing a given set of integers, and these integers are the second component of `ForcedIntegers` (PF). For every irreducible element in the list, we then remove those minimal generators fulfilling condition (4) in Proposition A.2. We add to our list of semigroups the semigroups obtained in the preceding step for which condition (5) holds, and then we proceed recursively.

EXAMPLE A.3. Let us illustrate the above procedure with $\mathsf{PF} = \{10, 13\}$. The number of irreducible numerical semigroups with Frobenius number 13 is 8. However, if we first call `ForcedIntegers` (the names we use in our implementation are slightly different), we get

```
gap> ForcedIntegersForPseudoFrobenius([10,13]);
[ [ 1, 2, 3, 5, 6, 10, 13 ], [ 0, 7, 8, 11, 12, 14 ] ]
```

Since we have modified the function `IrreducibleNumericalSemigroupsWithFrobeniusNumber` to output only those irreducible numerical semigroups containing the set $\{0, 7, 8, 11, 12, 14\}$, we obtain only two irreducible numerical semigroups: $S_1 = \langle 4, 7, 10\rangle$ and $S_2 = \langle 7, 8, 9, 10, 11, 12\rangle$.

For S_1, the only minimal generator that fulfills the conditions in Proposition A.2 is 10. If we remove 10 from S_1, we obtain $T_1 = \langle 4, 7, 17\rangle$, which already has the desired set of pseudo-Frobenius numbers.

As for S_2, again 10 is the only minimal generator fulfilling the conditions in Proposition A.2, and we obtain $T_2 = \langle 7, 8, 9, 11, 12\rangle$. This semigroup has pseudo-Frobenius numbers set equal to PF, and so, as with T_1, we do not need to look for new minimal generators to remove.

Thus, the only numerical semigroups with pseudo-Frobenius numbers set $\{10, 13\}$ are T_1 and T_2.

Appendix B. *Picking a single semigroup with given pseudo-Frobenius numbers*

Sometimes one may just be interested in obtaining one numerical semigroup with PF as set of pseudo-Frobenius numbers, or simply if $\mathcal{S}(\mathsf{PF}) \neq \emptyset$. Algorithm 3 may be too slow (it gives much information that will not be used). One could adapt the algorithm to stop once it encounters the first semigroup, but the information had to be transmitted recursively and one would end up with a slow algorithm. Next we propose an alternative (Algorithm 4). Instead of calling the recursive function, it tries to guess a path that leads to a leaf corresponding to a numerical semigroup. The procedure starts by choosing a random free integer v and tests its non-admissibility (by checking whether `SimpleForcedIntegers` returns fail when called with v as if it was forced). If one does not conclude that v is non-admissible, it is assumed to be a forced integer; and one continues while there are free integers. As an option the user can specify the maximum number of attempts. Its usage is illustrated in Examples B.1 and B.2.

Input : PF
Output: A numerical semigroup S such that $\mathtt{PF}(S) = \mathsf{PF}$ if it discovers some, fail if it discovers that no semigroup exists. Otherwise, it is suggested to use more attempts

```
if PF = {frob} or PF = {frob/2, frob} then
  Use [1, Proposition 2.7];
f_ints := ForcedIntegers (PF);
if f_ints = fail then
  return fail;
free := {1, ..., frob} \ (f_ints[1] ∪ f_ints[2]);
for i ∈ [1..max_attempts] do
    while free ≠ ∅ do
        v := an element chosen randomly in free;
        nfig := SimpleForcedIntegers ( f_ints[1] ∪ {v},f_ints[2]);
        nfie := SimpleForcedIntegers (f_ints[1],f_ints[2] ∪ {v});
        if nfig ≠ fail then
            if nfig[1] ∪ nfig[2] is an interval then
                return NumericalSemigroupByGaps (nfig[1]);
            else
                free := {1, ..., frob} \ (nfig[1] ∪ nfig[2]);
        else
            if nfie ≠ fail then
                if nfie[1] ∪ nfie[2] then
                    return NumericalSemigroupByGaps (nfie[1]);
                else
                    free := {1, ..., frob} \ (nfie[1] ∪ nfie[2]);
            else
                break;
▷ Info: Increase the number of attempts ...
```

Algorithm 4: ANumericalSemigroupWithPseudoFrobeniusNumbers.

It may happen that no semigroup has the given set as set of pseudo-Frobenius elements, and thus the output will simply be fail. The following example is meant to illustrate that one can use Algorithm 4 even for quite large Frobenius numbers. (The reader may obtain a different output since it depends on a random seed.)

EXAMPLE B.1. We look for a numerical semigroup with $\mathsf{PF} = \{100, 453, 537, 543\}$. The first execution of the function yields[†]

```
gap> pf := [ 100, 453, 537, 543 ];;
gap> ns := ANumericalSemigroupWithPseudoFrobeniusNumbers(pf);;time;
 MinimalGeneratingSystem(ns);
2440
[ 66, 94, 106, 126, 166, 184, 194, 206, 209, 216, 224, 230, 235,
 246, 256, 263, 267, 284, 295, 309, 363, 374, 379, 385, 391, 413 ]
```

[†]In [4], `ANumericalSemigroupWithPseudoFrobeniusNumbers` is called `RandomNumericalSemigroupWithPseudoFrobeniusNumbers`; in the next version of `numericalsgps` we will use the name employed in these examples (see also the development version of the package in https://bitbucket.org/gap-system/numericalsgps).

EXAMPLE B.2. If one of the free integers can neither be a gap nor an element, no semigroup exists:

```
gap> pf :=[30, 104, 118, 147, 197, 292, 298, 315, 333, 384, 408];;
gap> ns :=ANumericalSemigroupWithPseudoFrobeniusNumbers(
>  rec(pseudo_frobenius := pf, max_attempts := 100));time;
fail
22
```

References

1. V. BLANCO and J. C. ROSALES, 'The tree of irreducible numerical semigroups with fixed Frobenius number', *Forum Math.* 25 (2013) 1249–1261.
2. H. BRESINSKY, 'Symmetric semigroups of integers generated by 4 elements', *Manuscripta Math.* 17 (1975) 205–219.
3. M. DELGADO, 'intpic, a GAP package for drawing integers', http://www.gap-system.org/.
4. M. DELGADO, P. A. GARCÍA-SÁNCHEZ and J. MORAIS, 'NumericalSgps, a GAP package for numerical semigroups', version 1.0.1, 2015, http://www.gap-system.org/.
5. The GAP group, 'GAP – Groups, algorithms, programming', version 4.7.7, 2015, http://www.gap-system.org/.
6. J. KOMEDA, 'On the existence of Weierstrass points with a certain semigroup generated by 4 elements', *Tsukuba J. Math.* 6 (1982) 237–270.
7. A. M. ROBLES-PÉREZ and J. C. ROSALES, 'The genus, the Frobenius number, and the pseudo-Frobenius numbers of numerical semigroups with type two', Proc. Roy. Soc. Edinburgh Sect. A, to appear. Q3
8. J. C. ROSALES and P. A. GARCÍA-SÁNCHEZ, *Numerical Semigroups*, Developments in Mathematics 20 (Springer, 2010).
9. 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 (2004) 301–313.

M. Delgado
CMUP
Departamento de Matemática
Faculdade de Ciências
Universidade do Porto
Rua do Campo Alegre 687
4169-007 Porto
Portugal

mdelgado@fc.up.pt

A. M. Robles-Pérez
Departamento de Matemática Aplicada
Universidad de Granada
18071 Granada
Spain

arobles@ugr.es

P. A. García-Sánchez
Departamento de Álgebra
Universidad de Granada
18071 Granada
Spain

pedro@ugr.es

Please do not answer on this page but find the appropriate point in the text and make your annotation there.

Author Queries

Journal:	JCM
Article id:	00006
Author:	M. Delgado, P. A. García-Sánchez and A. M. Robles-Pérez
Short title:	Numerical semigroups with pseudo-Frobenius numbers

Q1 (Page 1)

Au: The distinction between surnames can be ambiguous, therefore to ensure accurate tagging for indexing purposes online (eg for PubMed entries), please check that the highlighted surnames have been correctly identified, that all names are in the correct order and spelt correctly.

Q2 (Page 1)

Au: Please check and confirm the shortened running is OK, or suggest a suitable alternative (fewer than 50 characters, including spaces).

Q3 (Page 20)

Au: Please update Ref. [7], if possible.