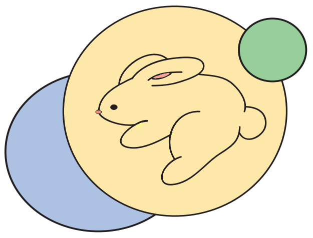

An Introduction to HASE = **H**oles in Pointed **A**ffine **Se**migroups

Florian Kohl Yanxi Li Johannes Rauh Ruriko Yoshida

August 12, 2016

Contents

1 Preface

HASE is a program that computes holes of pointed, affine semigroups. It is an implementation of the algorithm described by Hemmecke, Yoshida, and Takemura, for more information see [Hemmecke et al., 2009]. The program itself is written in Python 3, but it internally uses (and hence is dependent on) Normaliz[Bruns et al.], Macaulay2 [Grayson and Stillman], and on 4ti2 [4ti2 team, 2008]. The dependence on 4ti2 however is not strict, as there is the –Nsolve option avoiding 4ti2.

2 The Basics

2.1 Theoretical Background

This subsection is taken from [Kohl et al., 2016].

Let $A \in \mathbb{Z}^{m\times n}$ be a matrix with integral entries. Moreover, let $\mathbb{Z}_+ = \{0, 1, 2, 3, \dots\}$ be the set of nonnegative integers and let $\mathbb{R}_+ = [0, \infty)$. Then the semigroup generated by the columns $\boldsymbol{a}_1$, $\boldsymbol{a}_2$, ..., $\boldsymbol{a}_n$ of A is the set

$$Q = Q(A) = \{\mathbf{a}_1 x_1 + \cdots + \mathbf{a}_n x_n \mid x_1, \dots, x_n \in \mathbb{Z}_+\}. \tag{1}$$

Let $K = K(A)$ be the cone generated by the columns $\mathbf{a}_1, \dots, \mathbf{a}_n$ of A, i. e.

$$K = K(A) = \{\mathbf{a}_1 x_1 + \cdots + \mathbf{a}_n x_n \mid x_1, \dots, x_n \in \mathbb{R}_+\}.$$

The lattice $L = L(A)$ generated by the columns $\mathbf{a}_1, \ldots, \mathbf{a}_n$ of A is given by

$$L = L(A) = \{\mathbf{a}_1 x_1 + \cdots + \mathbf{a}_n x_n \mid x_1, \ldots, x_n \in \mathbb{Z}\}\,.$$

The semigroup $Q_{\text{sat}} = K \cap L$ is called the *saturation* of the semigroup Q with respect to the lattice L. It follows that $Q \subset Q_{\text{sat}}$ and we call Q *saturated* if $Q = Q_{\text{sat}}$ (this is also called *normal*). We call $H = Q_{\text{sat}} \setminus Q$ the set of *holes* of the semigroup Q. A hole $h \in H$ is *fundamental* if there is no other hole $h' \in H$ such that $h - h' \in Q$. Note that in contrast to H, F is always finite as it is contained in the fundamental parallelepiped, see [Takemura and Yoshida, 2008]. Let us illustrate these definitions in a small example.

Example 2.1 (The Frobenius Problem)**.** *Let p, q be two positive, coprime integers. In the notation above, this means $A = (p, q) \in A^{1\times 2}$. The semigroup generated by the columns of A is*

$$Q(A) = \{n \in \mathbb{Z}_+ | \, n = ap + bq\,,\ \textit{where}\ a,\ b \in \mathbb{Z}_+\}\,.$$

*It is a direct consequence of Bezout's theorem that the lattice generated by p and q is $L(A) = \mathbb{Z}$. However, if p and q are **not** coprime, then $L(A) \neq \mathbb{Z}$. Furthermore, we have that $K(A) = \mathbb{R}_+$. The saturation $Q_{sat} = K(A) \cap L(A) = \mathbb{R}_+ \cap \mathbb{Z} = \mathbb{Z}_+$. Thus, the set of holes $H = Q_{sat} \setminus Q$ is the of nonnegative integers that cannot be written as a positive, linear, integral combination of p and q. It is a theorem that there are only* finitely *many holes in $Q(A)$. The Frobenius problem asks to find the largest number that cannot be written as such a combination of p and q. This number is called the **Frobenius number** and it is denoted $g(p, q)$. Moreover, it can be shown that*

$$g(p, q) = pq - p - q$$

for a proof this theorem, see [Beck and Robins, 2007, Theorem 1.2, p. 6].

If we now choose $p = 3$ and $q = 7$, we can see that

$$H = \{1, 2, 4, 5, 8, 11\}.$$

and

$$F = \{1, 2\}.$$

2.2 A First Example

Let us now turn to an explicit computation of holes in an affine semigroup. Let $Q(A)$ be the semigroup generated by $(1, 0)^t$, $(1, 2)^t$, $(1, 3)^t$, and $(1, 4)^t$. This means that

$$A = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 2 & 3 & 4 \end{pmatrix}.$$

To compute the holes with HASE, we need to create an input-file *example.mat* which looks like

```
2 4
1 1 1 1
0 2 3 4
```

The first line specifies the dimension of the matrix and the other lines is the matrix A. We have to make sure that this file is in the same folder as Normaliz and ZSolve, which is an application from 4ti2. Now we can run the program by entering

```
python3 hase.py --normaliz "./normaliz" -v example
```

into the terminal. The output looks like

```
Using temporary directory /tmp/tmp080qluh7/
Read example.mat.  4 generators in dimension 2.
>>>>> Running normaliz to compute the fundamental holes.
Normaliz found 1 fundamental holes.
Looking at hole [1 1]
>>>>> Running zsolve to find the minimal solutions to f = Am - Al.
>>>>> Running M2 to compute the standard pairs.  Output:
    1: {x0}
```

This output means that there is one fundamental hole $(1,1)^t$ and H can be described as

$$H = \left\{(1,1)^t + m(1,0)^t \colon m \in \mathbb{Z}_+\right\}.$$

As mentioned in the preface, there is the option –Nsolve, which uses Normaliz to determine the integer solutions instead of using 4ti2. The command now looks like

```
python3 hase.py --normaliz  "./normaliz" --Nsolve "./normaliz" -v beispiel
```

The output now is

```
Read beispiel.mat.  4 generators in dimension 2.
>>>>> Running normaliz to compute the fundamental holes.
Normaliz found 1 fundamental holes.
Looking at hole [1 1]
>>>>> Running Normaliz to find the minimal solutions to f = Am - Al.
>>>>> Running M2 to compute the standard pairs.  Output:
    1: {x0}
```

2.3 Frobenius Revisited

Let's now turn to the Frobenius problem. We will start with a very small example to nicely interpret the output. With the notation of Example 2.1, we set $p = 3$ and $q = 5$. Our input file *frobenius.mat* should look like:

```
2 2
3 5
0 0
```

For technical reasons, we had to embed the problem into $\mathbb{R}^2$. The output is now:

```
Using temporary directory /tmp/tmp6qedfn44/
Read frobenius.mat.  2 generators in dimension 2.
>>>>> Running normaliz to compute the fundamental holes.
Normaliz found 2 fundamental holes.
Looking at hole [1 0]
>>>>> Running zsolve to find the minimal solutions to f = Am - Al.
>>>>> Running M2 to compute the standard pairs.  Output:
    1: {}
    x0: {}
      2
    x0  : {}
Looking at hole [2 0]
>>>>> Running zsolve to find the minimal solutions to f = Am - Al.
>>>>> Running M2 to compute the standard pairs.  Output:
    1: {}
    x1: {}
```

There are only two fundamental holes, but there are several standard pairs. The $x_i's$ in the output refer to the $(i-1)^{\text{st}}$ generator. For the hole $(1,0)$ this simply means, we have the holes $(1,0)$, then the hole $(1,0)+(3,0)$ and $(1,0)+2\cdot(3,0)$. Moreover, we have the holes with base point $(2,0)$, namely $(2,0)$ and $(2,0)+(5,0)$. Note that the list of holes is redundant, as the hole $(7,0)$ appears with two different base points. In general, i.e. in higher dimensions, we cannot expect a finite set of holes, so based on every hole there will be a monoid. This monoid is trivial in our case, which explains the empty brackets after the variables.

2.4 A Bigger Example

Now let us turn to a more complex problem. Let $A \in \mathbb{Z}^{9\times 16}$ be the matrix that fixes the row-, and column sums and that fixes the diagonal of a given 4×4 matrix, i.e.

$$A = \begin{pmatrix} 1&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0\\ 0&1&0&0&0&1&0&0&0&1&0&0&0&1&0&0\\ 0&0&1&0&0&0&1&0&0&0&1&0&0&0&1&0\\ 0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&1\\ 1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0\\ 0&0&0&0&1&1&1&1&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0&1&1&1&1&0&0&0&0\\ 0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1\\ 1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1 \end{pmatrix}.$$

Let us determine the holes of the semigroup generated by the columns of A. We have to save the matrix A in the file 44.mat:

```
9 16
1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0
0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0
0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0
0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1
1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1
1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1
```

Now we can run our program with the following command:

```
python3 hase.py --normaliz "./normaliz" -v 44
```

The output will be:

```
Normaliz found 6 fundamental holes.
Standard pairs of [0 0 1 1 0 0 1 1 1]:
    1: {x10, x11, x14, x15}
    1: {x0, x5, x10, x15}
Standard pairs of [0 1 0 1 0 1 0 1 1]:
    1: {x5, x7, x13, x15}
    1: {x0, x5, x10, x15}
```

```
Standard pairs of [0 1 1 0 0 1 1 0 1]:
   1: {x5, x6, x9, x10}
   1: {x0, x5, x10, x15}
Standard pairs of [1 0 0 1 1 0 0 1 1]:
   1: {x0, x5, x10, x15}
   1: {x0, x3, x12, x15}
Standard pairs of [1 0 1 0 1 0 1 0 1]:
   1: {x0, x5, x10, x15}
   1: {x0, x2, x8, x10}
Standard pairs of [1 1 0 0 1 1 0 0 1]:
   1: {x0, x5, x10, x15}
   1: {x0, x1, x4, x5}
```

For more about the fundamental holes of $Q(A)$ and an explicit description of the Hilbert basis of $Q(A)$, see [Kohl et al., 2016].

3 More Options

In this section, we want to give a very brief overview over the different options. Enter

```
python3 hase.py --normaliz "./normaliz" --help
```

in your terminal to see all possible flags.

```
usage: hase.py [-h] [-k] [--M2 M2] [--normaliz NORMALIZ]
                       [--zsolve ZSOLVE] [-t] [--time]
                       [--lp_solve LP_SOLVE] [-v]
                       filename

Compute the hole monoids of an affine semigroup.

positional arguments:
  filename              the name of the file containing the generators,
                        without the ending ".mat" (as columns of a matrix, in
                        4ti2 .mat format)

optional arguments:
  -h, --help            show this help message and exit
  -k, --keep-temporary-files
                        with this option, temporary files are created in the
                        same directory (using FILENAME-him as a basename) and
                        are not deleted when the program finishes. Note that
                        the temporary files for different fundamental holes
                        get the same name, so only the temporary files used
                        when computing the monoid of the last fundamental hole
                        are preserved.
  --M2 M2               the command to call Macaulay2 (including the path)
  --normaliz NORMALIZ   the command to call normaliz (including the path)
  --zsolve ZSOLVE       the command to call zsolve (including the path)
  --Nsolve              the command to use Nsolve instead of zsolve (including path)
  -t, --trick           for each fundamental hole f and generator g, check if
                        f+g is a hole. Only compute the hole monoid among the
                        remaining generators.
  --time                time each zsolve command
  --lp_solve LP_SOLVE   the command to call lp_solve (including the path)
```

```
-v, --verbose          increase output verbosity. Use several times ("-vv")
                       to further increase output verbosity
```

References

4ti2 team. 4ti2—a software package for algebraic, geometric and combinatorial problems on linear spaces. Available at www.4ti2.de, 2008.

Matthias Beck and Sinai Robins. *Computing the Continuous Discretely*. Undergraduate Texts in Mathematics. Springer, New York, 2007. ISBN 978-0-387-29139-0; 0-387-29139-3. Integer-point enumeration in polyhedra.

W. Bruns, B. Ichim, T. Roemer, R. Sieg, and C. Soeger. Normaliz. algorithms for rational cones and affine monoids. Available at `https://www.normaliz.uni-osnabrueck.de`.

Daniel R. Grayson and Michael E. Stillman. Macaulay2, a software system for research in algebraic geometry. Available at `http://www.math.uiuc.edu/Macaulay2/`.

R. Hemmecke, A. Takemura, and R. Yoshida. Computing holes in semi-groups and its applications to transportation problems. *Contributions to Discrete Mathematics*, 4(1):81 – 91, 2009.

Florian Kohl, Yanxi Li, Johannes Rauh, and Ruriko Yoshida. Semigroups — a computational approach. *preprint*, 2016. URL `http://arxiv.org/abs/1608.03297`.

A. Takemura and R. Yoshida. A generalization of the integer linear infeasibility problem. *Discrete Optimization*, 5(1):36–52, 2008.