HUHFA User Manual

Olga Heismann* Achim Hildenbrandt† Francesco Silvestri†

August 30, 2013

1 Introduction

The polyhedral approach to combinatorial optimization problems studies the structure of their associated polytopes. One way is to compute complete linear descriptions of small polytopes in order to generalize the equations and inequalities. "Small polytopes" might actually not look so small at first sight: There is often a huge number of facet-defining inequalities already for very small problem sizes.

However, there are also often many symmetries implied by the combinatorial structure of the problem which can be used to classify the facets. These symmetries act on the feasible solutions and naturally form a group. In their representation as maps on the variable values they can be extended to symmetries acting on the polytope, and one can prove that they map vertices of the polytope to vertices of the polytope, and facets to facets. We say that those facet-defining inequalities which are similar in the sense that they can be transformed onto each other by some symmetry belong to one class.

Understanding all the facet-defining inequalities of a combinatorial optimization problem polytope then reduces to understanding one facet from each class.

To do this classification, one applies the symmetries to the facet-defining inequalities and then checks whether any two facets can be transformed into each other and hence belong to the same class. Often, this check is not so easy as two linear expressions describing the same facet might differ by the sum of multiples of several equalities from the problem description.

The check can be accomplished by defining a so-called normal form for the representation of inequalities—inequalities which have the same normal form describe the same facet. To this end, problem-specific normal forms were developed for some extensively studied combinatorial optimization problems. In general, the representation of facet-defining inequalities in the orthogonal complement of the linear subspace spanned by the equations can be of course used as a normal form for the facets of a polytope. However, this needs

*Zuse Institute Berlin, Takustraße 7, 14195 Berlin, Germany, heismann@zib.de.

†Ruprecht-Karls-Universität Heidelberg, INF 368, 69120 Heidelberg, Germany, achim.hildenbrandt@informatik.uni-heidelberg.de, f.silvestri@stud.uni-heidelberg.de.

techniques from linear algebra and can therefore raise numerical issues. Unfortunately, normal forms that can be described combinatorially are often not known. Hence, having a method that can be applied to every combinatorial optimization problem and relies solely on the combinatorial structure of the polytope is desirable.

Indeed, HUHFA uses a novel technique for classifying facets without using normal forms. The main idea is to identify every facet defining inequality with the vertices of the polytope which satisfy it with equality. With this method, complete descriptions of polytopes computed by a software like PORTA[1] (or a similar package) can be analyzed to divide the facets into equivalence classes according to groups generated by given symmetry mappings.

For more information on the theoretical background of how HUHFA works see the article downloadable from the HUHFA website[2].

[1] `http://comopt.ifi.uni-heidelberg.de/software/PORTA`

[2] `http://comopt.ifi.uni-heidelberg.de/people/hildenbrandt/HUHFA`

2 Installing

In order to use HUHFA, the first thing you need to do is to move the following files in one directory:

- classify.cpp
- input.cpp
- output.cpp
- synthesize.cpp
- script.cpp
- huhfa.h

Afterwards, the file *script.cpp* has to be build with a given C++ compiler (you may alternatively use the provided makefile). In order to be independent of operating system, we will refer to the resulting file as *huhfa*.

3 Input Formats

In order for HUHFA to work properly, three files are needed to encode the information of a given polytope. These files need to have the same name with different filename extensions *.huh*, *.poi* and *.poi.ieq*, so we that we can refer to a complete set of input files

- **polytope.huh**
- **polytope.poi**
- **polytope.poi.ieq**

just by "polytope" where the output would be stored in the file

- **polytope.huhfa**.

The format of the input files is described in the following subsections, each followed by an example describing the three-dimensional unit-cube $Q_3 = [0,1]^3$. Note that HUHFA uses space characters in order to parse the input files. Therefore, it is necessary to use at least one space character whenever it is used in the following descriptions.

polytope.poi

- Line 1 denotes **the dimension of the modeling space**, that is, the number of variables used, by $DIM = n$.
- Line 2 consists of the word *INDEX*.
- Line 3 contains all variables used in the modeling process, separated by a space character. Note that variable names can be arbitrary as long as the **first character is a letter** and it **contains no spaces**.
- Line 4 consists of the word *CONV_SECTION*.
- Lines $5,\ldots,4+v$ each contain the coordinates of a single vertex from the given polytope, separated by a space character, in the order of the variables in Line 3.
- Line $5+v$ consists of the word *END*.

This way, the file *Q3.poi* of Q_3 is given by:

```
1  DIM = 3
2  INDEX
3  x1 x2 x3
4  CONV_SECTION
5  0 0 0
6  1 0 0
7  0 1 0
8  0 0 1
9  0 1 1
10 1 0 1
11 1 1 0
12 1 1 1
13 END
```

polytope.poi.ieq

This file contains facet-defining inequalities describing all facets of the polytope.

- Line 1 contains $DIM = n$ where n is the number of variables used, just as in **Polytope.poi**.
- Line 2 consists of the word *INDEX*.
- Line 3 contains the variables used in the same format and order as in **Polytope.poi**.
- Line 4 consists of the word *INEQUALITIES_SECTION*.
- Lines $5,\ldots,4+f$ contain a facet defining inequality each.
- Line $5-f$ consists of the word *END*.

Suppose a facet-defining inequality is given by $a^T x \leq b$ where $a = (a_i)$, $b = (b_i)$ and $x = (x_i)$ are the variables from Line 3. We set $a_i = s_i \tilde{a}_i$ where s_i denotes the sign of a_i and $\tilde{a}_i$ is the absolute value of a_i. Then the inequality is expressed as the line

$$\text{(no.) } s_1 a_1 x_1 s_2 a_2 x_2 \ldots s_n a_n x_n \quad <= \quad \text{b}$$

Please note the following:

- The brackets at the beginning will not be parsed and can be used to number the inequalities. They can also be left out. Either way, at least a space character has to come before the first term of the inequality.
- At least a space character before and after <= is required.
- There may be space characters after the variables, but the terms $s_i a_i x_i$ may not be separated.
- Any terms where $a_i = 0$ may be dropped.
- Any terms where $\tilde{a}_i = 1$ may drop the $\tilde{a}_i$.

For example, consider the facet-inducing inequality $x1 \leq 1$ for Q_3. Following the guidelines above, HUHFA can parse all of the following lines in order to read this inequality:

```
1 (44) +1x1+0x2+0x3 <= 1
2 (44) +x1 +0x3 <= 1
3 +1x1 +0x2+0x3 <= 1
4 +x1 <= 1
```

Therefore, *Q3.poi.ieq* could have the following content:

```
1  DIM = 3
2  INDEX
3  x1 x2 x3
4  INEQUALITIES_SECTION
5  ( 1) +x1 <= 1
6  ( 2) +x2 <= 1
7  ( 3) +x3 <= 1
8  ( 4) -x1 <= 0
9  ( 5) -x2 <= 0
10 ( 6) -x3 <= 0
11 END
```

polytope.huh

Each line of this file has the same structure and describes a bijection of the vertices in **polytope.poi** given through an affine map.

Suppose such a bijection is given by $x \mapsto Ax + b$ where $A = (a_{ij})$, $b = (b_i)$ and $x = (x_i)$ contains the names of the variables in the same order as in **polytope.poi**. Further, suppose that $a_{ij} = s_{ij}\tilde{a}_{ij}$ where s_{ij} is the sign of a_{ij} and $\tilde{a}_{ij}$ describes the absolute value of a_{ij}, analogously $b_i = t_i\tilde{b}_i$. Then the corresponding line in **polytope.huh** looks like this:

$$s_{11}\tilde{a}_{11}x_1 s_{12}\tilde{a}_{12}x_2 \ldots s_{1n}\tilde{a}_{1n}x_n t_1\tilde{b}_1 \;\; s_{21}\tilde{a}_{21}x_1 s_{22}\tilde{a}_{22}x_2 \ldots s_{2n}\tilde{a}_{2n}x_n t_2\tilde{b}_2 \;\; \ldots \;\; s_{n1}\tilde{a}_{n1}x_1 \, s_{n2}\tilde{a}_{n2}x_2 \ldots s_{nn}\tilde{a}_{nn}x_n t_n\tilde{b}_n \; ;$$

In essence, we write down each coordinate of the image as a string of the corresponding term, separate the coordinates by the space character and finish with a space character followed by a semicolon.

Please note:

- The absolute value $t_i\tilde{b}_i$ has to be the last summand in every coordinate.
- Any terms where $a_{ij} = 0$ or $b_i = 0$ may be dropped.
- Any terms where $\tilde{a}_{ij} = 1$ may drop the $\tilde{a}_{ij}$.

To make this more clear, consider the map given by

$$(x1, x2, x3) \mapsto (1 - x1, x3, x2).$$

When using this map on the vertices of Q_3, it is clear that the first entry is "flipped" and the other two entries are permuted, so this certainly is a bijection of the vertices of Q_3 and thus valid for *Q3.huh*. This map is affine and thus has a representation in the form above which is

$$x \mapsto \begin{pmatrix} -1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} x + \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$$

Therefore, *Q3.huh* could contain either of the following lines:

```
1 -1x1+0x2+0x3+1 0x1+0x2+1x3+0 0x1+1x2+0x3+0 ;
2 -x1+1 +x3 +x2 ;
```

The whole file *Q3.huh* could look like this:

```
1 -x1+1 +x3 +x2 ;
2 +x3 -x2+1 +x1 ;
3 +x2 +x1 -x3+1 ;
```

4 Usage

Prior to using *huhfa*, make sure that the files **polytope.huh**, **polytope.poi** and **polytope.poi.ieq** we want to classify are in one directory.

You can start *huhfa* from a terminal by using the following parameters:

- huhfa **polytope**
- huhfa **polytope** *rep*
- huhfa **polytope** *closed*
- huhfa **polytope** *repclosed*

In this case, **polytope** simply denotes the name of the files we want to process and the second argument is optional.

Without a second argument, *huhfa* will classify the facets with regard to the given symmetries and store the complete equivalence classes in **polytope.huhfa**. Note that in order to do this, *huhfa* will implicitly compute the closure of the vertex-bijections given by **polytope.huh**.

Using *rep* will only affect the output file **polytope.huhfa**. Instead of the whole equivalence classes, only a representative facet-defining inequality will be stored for each equivalence class.

Using *closed* may only be used when the maps given in **polytope.huh** are already closed under composition. This will result in a speedup as the computation of the closure can be skipped, but will most certainly result in errors and/or wrong results when the maps in **polytope.huh** are not closed.

Using *repclosed* is used in order to use *rep* and *closed* at the same time.

So in order to classify our example with the standard settings, we would enter

huhfa Q3

into the terminal.

5 Descriptions of Available Methods

The central part of HUHFA is the class *huhfa* which is used for all computations. *huhfa* has the following methods:

- *huhfa(string source)*
  The constructor is reading the input files and stores their content as the raw data of the polytope.

- *huhfa.synthesize()*
  This method uses the raw data given by the input files and computes the data needed for the actual classifying process. This includes creating incidence vectors as well as transforming the affine maps into vertex permutations. The transformation reduces the time of the following operations.

- *huhfa.classify(bool complete)*
  This method computes the equivalence classes using the synthesized data.

- *huhfa.output(string result, bool rep)*
  This is used to store the equivalence classes into an output file.

It should be clear from context that the methods described above may only be used in the given order as they need the information of the method before to work properly.

Additionally, there are some test methods in order to inspect the internal data of the huhfa class. They are not intended to be used by the user but are self-explanatory for a programmer working with the huhfa-code and may thus be looked up in the code.