Computation of Hermite Normal Forms of Integer Matrices

Importance of HNF

Many Integer programming algorithms, or algorithms that compute with modules over Z, have computation of HNF as a core building block. Computation of HNF over integer matrices of large size and value [value meaning the size of the integer entries in the matrix] is therefore a very important problem to solve.

Definition adopted for HNF

There are various definitions for HNF which present the same idea for an HNF in different ways. Different algorithmic implementations choose to represent HNF in different ways will be stated when the corresponding algorithm is laid out. However, if not mentioned specifically, HNF is mostly the column reduced echelon form. For the naïve Gaussian procedure, I have assumed the lower triangular definition as described below.

For a mXn matrix A, with entries a[i][j]

1. a[i][j] = 0 if 1<=i<j<=n - Lower triangular
2. a[i][i] >=0 for 1<=i<=rank(A). - Diagonal Positive
3. 0<=a[i][j]<a[i][i] for 1<=j<i<=rank(A) - Row entries non negative and less than corresponding diagonal.

This definition was derived from this reference :
https://www.math.uwaterloo.ca/~wgilbert/Research/GilbertBricklaying.pdf (Page 2, Paragraph 2).

For other procedures, the definitions are described in their corresponding references.

Primitive Algorithm for Computing HNF

Gaussian Elimination to obtain a HNF

Issues/Problems –

1. Too slow for practical purposes on large matrices with large numbers. Time complexity is more than polynomial time.

Possible Improvements –

Use other algorithms optimized for large matrices and feasible to work with on large numbers. Kannan and Bachem provide a polynomial time solution but do not escape the entry explosion problem(or atleast, do not provide a good upper bound on the size of intermediate entries).

Computing the GCD and expressing it as a linear combination of its constituent elements can help reduce the corresponding row/column to 0 faster as described in the paper by Kannan and Bachem. This is why computing the GCD as a linear combination of its constituents is important. References and description below(in Stage I algorithms).

Stage I algorithms for Computing HNF

Kannan and Bachem

Time Complexity : - Polynomial Time

Reference : - PDF

Issues/Problems : -

Algorithm is fast enough but the upper bound on intermediate matrix entries is 20n^3 – Theorem 1 in reference

Possible Improvements : -

Chou and Collins algorithm improves on computation speed although even this algorithm cannot guarantee an upper bound less than 20n^3 on the intermediate entries in the matrix. Again, reference.

Blankinship Algorithm, as described below, helps reduce the entry explosion problem thus providing a reasonable upper bound on the size of the intermediate entries.

Stage II Algorithms for Computing HNF

These algorithms are better than Stage I in the sense that they have a good upper bound on the size of the intermediate integer entries while the HNF is computed. A particularly interesting Heuristic is that of the R1 matrix [a specific test case 26X27] which has a max intermediate entry of size 12(decimal digits) in Stage I algorithms while has only a 7(decimal digit) max intermediate entry if HNF is computed using Stage II algorithms.

Blankinship Algorithm

This algorithm is useful in efficiently computing the GCD of a n-dim vector expressed as a linear combination of the n elements of the vector. Formally, we get a unimodular Vector **P** such that for a vector **a.**

$$P = \sum_{i=1}^{N} p_i a_i = \gcd(a_1 \dots a_n) \quad \text{and} \quad \sum_{j=2}^{N} p_j a_j = 0$$

Improvements to this algorithm and usage of these to calculate the HNF form of a matrix could be used and implemented as they solve the problem of entry explosion as well as give good time performance. The process is illustrated in this reference.

Stage III Algorithms for Computing HNF

These are the current state of the art algorithms being used for computation of HNF. There are two of such algorithms which I could encounter.

1. **Micciancio and Warinschi**
2. **Pernet and Stein**

The Pernet and Stein Algorithm is a modified version of the Micciancio and Warinschi Algorithm and hence it is preferable that it be implemented with no need for implementation of Micciancio and Warinschi, as the algorithm is an an improvement of the same. This reference describes the algorithm in detail.

Most of the steps of this algorithm can be modularized , like the Add Row step, the Add Column Step as well as the Hermite Transition steps and the partial Hermite Computation Step. These modules can then be used together to couple and implement the Pernet and Stein Algorithm.

This Algorithm is beneficial in the fact that it has the stated best known time complexity for the problem it solves. The keyword Random can be found emphasized in the paper, meaning if the input matrix entries are not random and follow a pattern, this algorithm is probably not the best deal and other good algorithms might exist depending upon the nature of the pattern.

Stage IV Algorithms for Computing HNF

These are not Stage IV in the sense that these algorithms are better than the Stage III algorithms, but these are possible parallel implementations of the Stage I algorithms for Computing HNF. The procedures can be be found in this reference and this is something which can be included as a part of a stretch goal.