Sparse Matrix Methods and Probabilistic Inference Algorithms

Radford M. Neal

Dept. of Statistics and Dept. of Computer Science
University of Toronto

http://www.cs.utoronto.ca/~radford/
radford@stat.utoronto.ca

Part I

Faster Encoding for Low Density Parity Check Codes Using Sparse Matrix Methods

IMA Program on Codes, Systems and Graphical Models, 1999

The Parity Check Matrix

Suppose we will send blocks of N bits (0's and 1's) through a channel.

To be able to correct errors, we reduce the number of possible blocks by requiring that a block satisfy M parity checks.

We can express this by saying a valid block (or *codeword*) must satisfy

$$\mathbf{Hx} = \mathbf{0}$$

Here $\mathbf{x}$, the codeword, is a column vector of N bits, $\mathbf{0}$ is a column vector of N zeros, and $\mathbf{H}$ is an $M \times N$ *parity check matrix*, with $M < N$.

All arithmetic is done modulo 2 (equivalently, in GF(2)), where addition and subtraction are both XOR, and multiplication is AND.

The Encoding Problem

Let us assume that the rows of $\mathbf{H}$ are linearly independent. There will then be 2^{N-M} valid codewords, and we can use a codeword to uniquely represent a source block of $N-M$ bits.

The encoding problem: Define and compute a mapping from these $N-M$ source bits to the N bits of a codeword.

We will consider only *systematic* mappings, in which the $N-M$ source bits are directly represented by a subset of the N codeword bits. (The receiver can then easily find them.)

The other M bits of the codeword are chosen to satisfy the parity checks. We need to:

1) Choose which are the systematic source bits, and which are the parity check bits.

2) Figure out how to compute the M parity check bits given the $N-M$ source bits.

A Dense Encoding Method

Let's partition $\mathbf{H}$ into an $M \times M$ left part, $\mathbf{A}$, and an $M \times N$ right part, $\mathbf{B}$, after rearranging columns if necessary to make $\mathbf{A}$ non-singular.

Partition a codeword, $\mathbf{x}$, in the same way, into M check bits, $\mathbf{c}$, and $N-M$ source bits, $\mathbf{s}$.

The parity check equation, $\mathbf{Hx} = \mathbf{0}$, becomes

$$[\ \mathbf{A} \mid \mathbf{B}\ ] \left[\begin{array}{c} \mathbf{c} \\ - \\ \mathbf{s} \end{array} \right] = \mathbf{0}$$

From this, we get

$$\mathbf{Ac} + \mathbf{Bs} = \mathbf{0}$$

and hence

$$\mathbf{c} = \mathbf{A}^{-1}\mathbf{Bs}$$

We can pre-compute $\mathbf{A}^{-1}\mathbf{B}$, and then find the check bits $\mathbf{c}$ by multiplying the source bits $\mathbf{s}$ by this matrix. This takes time proportional to $M(N-M)$.

A Mixed Encoding Method

Suppose $\mathbf{H} = [\ \mathbf{A} \mid \mathbf{B}\ ]$ is sparse, and hence that $\mathbf{B}$ is as well. For LDPC codes, the number of 1's in a row of $\mathbf{B}$ will be constant, at least on average, independent of N.

It may then be faster to compute $\mathbf{c} = \mathbf{A}^{-1}\mathbf{B}\mathbf{s}$ in two steps:

1) Compute $\mathbf{z} = \mathbf{B}\mathbf{s}$, in time proportional to M, exploiting the sparseness of $\mathbf{B}$.

2) Compute $\mathbf{c} = \mathbf{A}^{-1}\mathbf{z}$, in time proportional to M^2.

The total time is of order M^2. This is better than the previous order $M(N-M)$ method when $M < N-M$ — ie, when the rate of the code is greater than 1/2.

We will next see how sparsity in $\mathbf{A}$ can be exploited as well.

Reduction to Upper Triangular Form

We can find $\mathbf{c} = \mathbf{A}^{-1}\mathbf{z}$ by using row operations to reduce $\mathbf{A}$ to an upper triangular matrix:

$$\begin{bmatrix} 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 0 & 1 \\ \underline{0} & 1 & 1 & \underline{0} \\ 0 & 0 & 1 & 0 \\ \underline{0} & 1 & 0 & \underline{1} \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & \underline{0} & \underline{1} & 1 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ \underline{1} \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & \underline{0} & 1 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \end{bmatrix}$$

Using *backward substitution*, we can now find that $c_4 = 1$, $c_3 = 0$, $c_2 = 1$, $c_1 = 1$.

Recording the Reductions in a Lower Triangular Matrix

The previous process reduced the equation $\mathbf{Ac} = \mathbf{z}$ to $\mathbf{Uc} = \mathbf{y}$, where $\mathbf{U}$ is upper triangular, and $\mathbf{y}$ was found as we reduced $\mathbf{A}$ to $\mathbf{U}$.

To solve $\mathbf{Ac} = \mathbf{z}$ for many $\mathbf{z}$ without going through the reduction process every time, we record how to find $\mathbf{y}$ as the solution of $\mathbf{Ly} = \mathbf{z}$, where $\mathbf{L}$ is lower triangular. This equation is easily solved by *forward substitution*.

For the example, we get:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

which can be solved to give $y_1 = 0$, $y_2 = 1$, $y_3 = 0$, $y_4 = 1$.

Putting it All Together

For the reduction to work, $\mathbf{A}$ must be non-singular, with rows and columns ordered to give 1's on the diagonal when needed.

We can find such a sub-matrix as follows:

Set $\mathbf{U}$ and $\mathbf{L}$ to all zeros.
Set $\mathbf{F}$ to $\mathbf{H}$.
for $i = 1$ to M:
  Find a non-zero element of $\mathbf{F}$ that is in row i, column i, or in a later row/column.
  Rearrange rows and columns of $\mathbf{F}$ and $\mathbf{H}$ from i onward to put this element in row i, column i.
  Copy column i of $\mathbf{F}$ up to row i to column i of $\mathbf{U}$.
  Copy column i of $\mathbf{F}$ from row i to column i of $\mathbf{L}$.
  Add row i of $\mathbf{F}$ to later rows with a 1 in column i.
end
Set $\mathbf{B}$ to the last $N-M$ columns of the rearranged $\mathbf{H}$.

We use $\mathbf{B}$, $\mathbf{L}$, and $\mathbf{U}$ to find parity checks for $\mathbf{s}$:

Compute $\mathbf{z} = \mathbf{B}\mathbf{s}$, exploiting the sparseness of $\mathbf{B}$.
Solve $\mathbf{L}\mathbf{y} = \mathbf{z}$ for $\mathbf{y}$ by forward substitution.
Solve $\mathbf{U}\mathbf{c} = \mathbf{y}$ for $\mathbf{c}$ by backward substitution.

Finding a Sparse LU Decomposition

We usually have a choice of non-zero elements to use next. We can use this freedom to try to make $\mathbf{L}$ and $\mathbf{U}$ as sparse as possible.

One strategy is the *minimal column* heuristic:

> Pick a non-zero element in row i or later from a column of $\mathbf{F}$ (from i onwards) that has the minimal number of non-zeros (but which does have a non-zero at row i or later).

This minimizes the number of non-zeros that will be immediately added to $\mathbf{L}$ and $\mathbf{U}$.

The *minimal product* heuristic is more forward looking:

> Pick the non-zero element from row i, column i or later that minimizes the product of
>
> - the number of non-zeros in its row minus 1
> - the number of non-zeros in its column (from row i on) minus 1.

This minimizes the number of modifications to other rows, which often produce non-zeros that are of later significance.

The Matrix $\mathbf{A}^{-1}\mathbf{B}$ *for a Rate* $1/2$ *LDPC Code with 3 Checks per Bit,* $M = 35$

0	0	0	0	0	0	0	0	0	0	#	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	#	0	#	0
0	0	0	0	#	0	0	0	0	0	0	#	0	0	0	0	0	0	0	0	0	0	0	0	0	#	0	0	0	0	0	0	0	0	0
#	0	0	0	0	0	0	0	0	0	#	#	#	0	0	#	0	0	0	0	0	0	0	#	#	#	#	0	0	0	0	0	#	#	0
0	0	#	0	0	0	0	0	#	0	#	0	0	0	0	0	0	#	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	#	#	#	#	0	#	#	#	#	0	0	#	0	#	#	0	#	#	#	0	0	#	#	#	0	#	#	0	#	#	#
#	0	#	0	#	0	#	0	#	0	#	0	0	#	0	0	#	#	0	0	#	0	0	0	0	0	0	0	0	0	0	0	0	#	0
#	0	0	0	#	0	#	0	0	0	0	0	0	0	0	0	#	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	#	0	#	0	#	0	#	0	0	#	#	#	#	0	#	#	#	0	#	#	0	#	0	#	0	#	#	#	0	0
#	0	0	#	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	#	#	0	0	0	0	0	0	0	0	0	0	0	0	0	0
#	#	0	0	#	#	0	0	#	0	0	#	0	#	0	#	#	#	0	#	#	#	0	#	0	#	0	#	0	0	0	#	#	0	#
#	0	0	0	0	#	#	#	0	0	0	0	#	#	0	0	#	#	#	0	#	#	#	0	0	#	#	0	0	0	#	#	0	0	#
0	#	0	0	0	0	0	0	#	0	0	0	0	#	0	0	0	0	#	#	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	#	#	0	#	#	#	0	#	#	0	0	#	#	#	0	#	0	#	0	0	#	#	#	0	#	#	0	#	0	0	#	#	0
0	0	#	0	0	#	#	0	#	#	0	#	#	0	#	0	0	#	#	0	#	0	0	0	0	0	0	#	#	#	0	#	#	0	0
0	0	#	0	0	#	#	#	#	0	#	#	0	#	0	#	#	0	#	#	#	#	#	#	#	0	0	0	0	0	#	0	#	0	#
0	0	#	0	0	#	#	0	0	#	#	0	#	0	0	#	0	#	#	0	0	0	0	#	#	0	#	#	0	0	#	0	#	#	0
#	#	0	0	#	#	0	0	0	#	0	#	0	#	#	#	0	0	#	#	#	#	#	#	#	#	#	#	0	0	0	#	#	0	0
0	0	#	0	0	0	#	#	#	0	0	0	#	0	0	#	#	0	#	#	0	#	0	0	0	0	#	#	0	#	#	0	0	0	0
0	0	#	0	#	#	#	#	0	#	0	#	#	0	#	#	0	#	0	#	0	#	0	0	#	#	0	#	#	#	0	#	0	0	#
#	0	0	0	#	#	#	#	0	0	#	0	0	0	#	0	#	#	0	0	#	#	0	#	0	0	#	#	#	#	0	#	#	#	#
0	#	#	#	#	#	#	#	0	0	#	0	0	#	0	0	0	#	0	#	#	#	0	#	#	#	#	0	#	#	0	0	#	#	#
#	0	#	0	#	0	0	0	#	0	#	0	0	#	0	0	#	#	0	0	#	0	0	0	0	0	0	0	0	0	0	#	#	#	0
0	0	#	0	0	#	#	0	0	#	#	0	0	0	#	0	0	#	0	#	0	0	#	#	#	0	#	#	0	#	0	0	#	#	0
0	0	#	0	#	0	#	#	#	0	0	#	#	0	0	#	#	0	#	#	0	#	#	0	0	#	#	#	#	#	#	#	0	0	0
#	0	#	0	0	#	0	0	#	0	#	0	#	0	0	0	0	#	0	#	0	0	0	#	#	#	#	#	#	0	#	#	0	#	0
#	0	0	0	0	0	0	0	0	#	#	#	#	0	0	#	0	0	0	0	#	0	0	#	#	#	#	0	0	0	0	0	#	#	0
0	0	0	#	0	#	0	0	0	#	#	0	0	0	0	#	#	#	0	#	#	#	0	#	#	#	0	#	#	#	#	0	#	#	0
0	0	#	0	0	0	0	0	#	0	#	#	#	0	0	0	0	0	0	0	#	0	0	0	#	0	#	#	0	#	#	#	0	#	#
#	#	0	0	#	#	0	0	0	0	0	#	0	#	0	#	#	#	0	#	#	#	#	#	#	#	#	#	0	0	0	#	#	0	#
0	0	0	#	0	0	0	0	0	0	#	#	0	0	#	0	0	0	0	0	0	0	0	0	0	#	#	0	#	0	#	0	#	0	0
0	0	#	0	0	#	#	0	#	#	0	#	0	0	0	#	0	#	#	0	#	0	0	0	0	0	0	#	0	#	0	#	#	0	0
0	0	#	0	0	0	#	0	#	0	#	#	#	#	#	#	#	#	#	#	0	0	#	0	#	0	#	#	#	#	#	#	0	#	0
0	#	#	0	0	0	#	#	0	#	0	0	#	#	0	#	#	0	0	0	0	0	0	0	0	0	#	#	0	0	#	0	0	0	0
#	0	0	0	0	#	#	#	0	0	0	0	#	#	0	#	#	#	#	0	#	#	#	0	0	#	#	#	0	0	0	#	#	0	#
0	#	0	0	#	#	#	#	0	#	#	#	#	0	0	#	0	#	#	0	#	0	#	#	0	#	#	#	0	#	#	0	#	#	#

The Matrix $\mathbf{A}^{-1}$ *for This Code.*

0	0	#	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
#	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	#	0	0	#	0	0	#	#	#	0	0	#	#	#	#	0	0	0	#	0	0	#	0	0	0	#	#	0	#
0	#	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
#	#	0	#	#	0	#	0	#	0	#	#	0	#	0	0	0	0	0	#	0	0	#	0	0	#	0	0	0	0	0	0	0	#	#
0	#	0	0	0	0	0	0	0	0	0	0	0	0	#	0	0	0	#	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	#	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	#	#	#	0	#	0	0	0	#	#	0	0	#	#	0	#	#	#	0	0	0	#	0	#	0	0	#	#	#	#	0	0	#	#
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	#	0	0	0	0	0	0	0	0
0	#	0	#	0	0	0	#	#	0	0	#	#	#	#	0	0	0	0	#	#	#	#	0	0	#	0	0	#	#	0	0	0	#	#
#	#	0	#	#	#	0	#	#	#	0	0	0	0	0	#	#	#	#	0	#	0	#	#	0	#	0	0	0	#	#	0	0	0	0
0	0	0	0	0	0	#	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	#	0	0	0	0	0	#	#	#	0	0	#	0	#	#	#	0	#	0	0	#	#	0	0	#	#	#	0	#	#	#	0	#	0
0	#	0	0	0	#	#	#	0	#	#	#	0	#	#	#	#	#	0	#	#	0	#	#	#	0	0	0	0	#	#	0	0	#	#
#	0	#	#	#	0	0	#	0	#	0	#	#	#	0	#	0	0	0	0	0	0	#	#	#	#	0	#	0	#	#	#	#	0	#
0	#	#	#	#	#	#	0	#	0	#	0	#	0	0	0	0	#	#	0	#	#	0	#	#	#	#	#	0	0	0	#	0	#	#
0	#	0	#	0	#	0	#	#	#	0	#	#	#	#	0	0	0	0	#	#	#	#	0	0	#	0	0	#	#	0	0	0	#	#
0	0	#	#	0	0	#	#	0	0	0	#	0	0	0	#	0	0	#	#	#	0	0	0	0	0	0	#	#	0	0	0	0	0	0
#	0	#	#	0	#	#	#	0	#	0	#	#	0	0	#	0	0	#	#	#	#	0	0	0	0	0	#	#	0	0	0	0	0	0
#	0	#	#	0	0	#	#	#	#	0	#	0	#	0	#	#	#	0	0	0	#	0	0	#	0	0	#	#	0	0	#	#	0	#
0	#	#	0	0	#	0	0	0	#	#	#	0	#	#	#	0	#	0	#	#	0	#	0	#	0	0	0	0	#	0	0	0	#	0
0	#	0	0	0	0	0	0	0	0	0	0	0	0	#	#	0	0	#	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	#	0	0	0	0	0	#	#	#	0	0	#	#	#	#	#	0	#	0	0	#	#	0	0	#	#	#	0	#	#	#	0	#	0
#	0	#	#	0	0	#	#	0	0	0	#	0	0	0	#	0	0	#	#	#	#	0	0	0	0	0	#	#	0	0	0	0	0	0
0	#	#	#	#	#	0	0	#	0	0	#	#	#	0	0	#	#	#	0	#	0	0	0	#	0	0	#	0	0	0	#	#	0	#
0	0	0	0	0	#	0	0	#	0	0	0	#	#	0	0	#	#	#	#	#	0	0	0	#	0	0	#	0	0	0	#	#	0	#
0	#	0	#	0	#	0	0	0	#	#	0	0	#	#	0	0	#	#	0	0	0	#	0	#	0	0	#	#	#	#	0	0	#	0
0	0	#	#	#	0	0	#	0	#	0	#	#	#	#	#	#	0	#	#	0	#	0	0	0	#	#	#	0	#	#	#	0	0	0
0	#	0	#	0	0	0	#	#	#	0	#	#	#	#	0	0	0	0	#	#	#	#	0	0	#	0	0	#	#	0	0	0	#	#
0	0	#	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	#
0	#	0	0	0	#	#	#	0	#	#	#	0	#	#	#	#	#	0	#	#	0	0	#	#	0	0	0	0	#	#	0	0	#	#
0	0	0	0	#	0	#	0	#	0	0	0	#	0	#	0	#	#	0	0	#	#	#	#	0	#	0	0	#	#	#	0	0	0	0
0	0	#	#	0	0	0	0	0	0	0	#	0	0	0	#	0	0	#	#	#	0	0	0	0	0	0	#	#	0	0	0	0	0	0
#	#	0	0	#	#	0	#	#	#	0	0	0	0	0	#	#	#	#	0	#	0	#	#	0	#	0	0	0	#	#	0	0	0	0
#	#	0	#	#	0	#	0	#	0	0	#	0	#	0	0	0	0	0	#	0	0	#	0	0	#	0	0	0	0	0	0	0	#	#

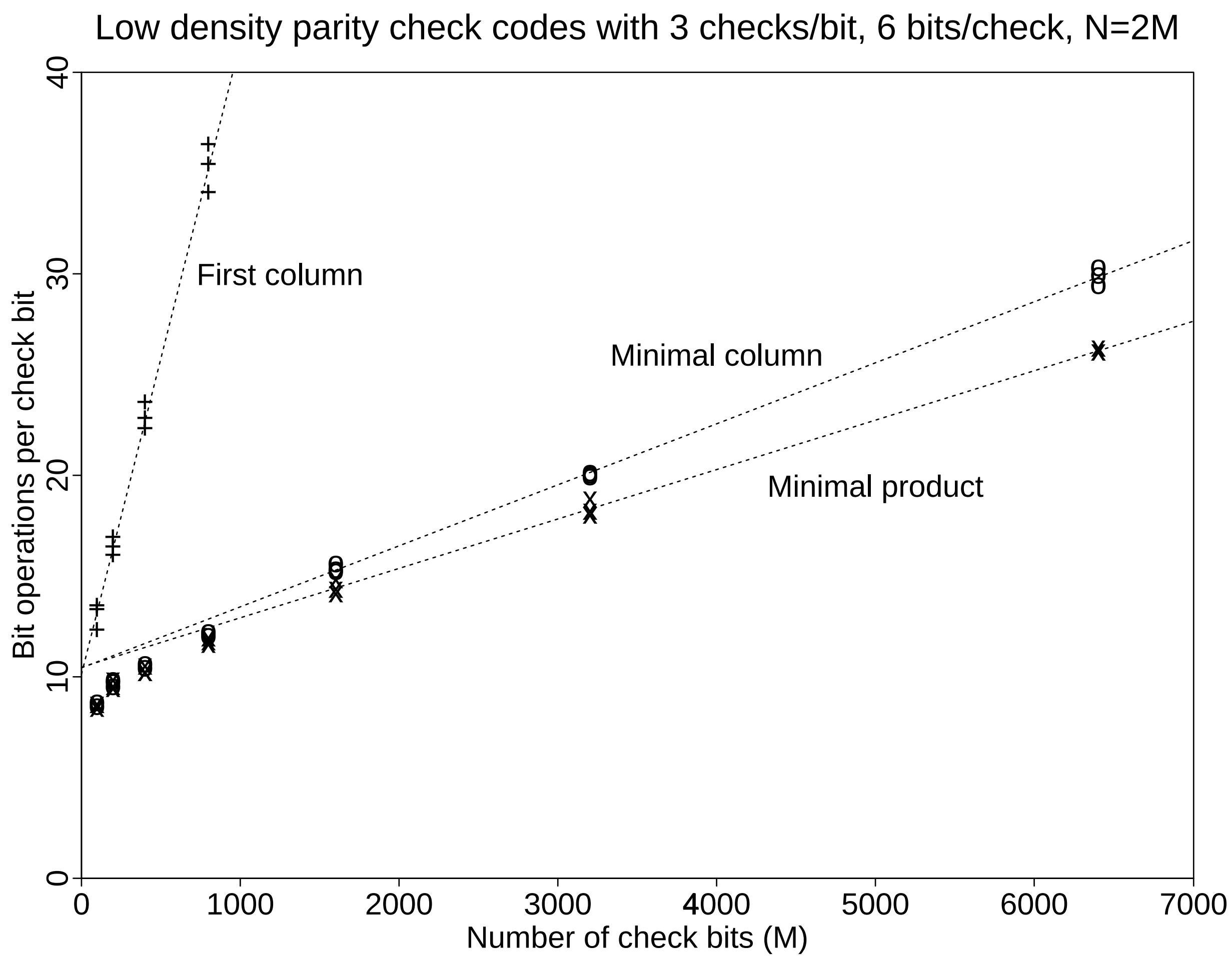
The $\mathbf{L}$ *Matrix for This Code, As Found With the Minimal Product Heuristic*

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 # 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 # 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 # 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 # 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 # 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 # 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 # 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 # 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 # 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 # 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 # 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 # 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 # 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 # 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 # 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 # 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 # 0 0 0 0 0 # 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 # 0 0 0 0 0 0 # 0 # 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 # 0 0 0 0 0 0 0 # 0 0 0 0 # 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 # 0 0 # 0 0 # 0 0 0 0 # 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 # 0 # # 0 0 0 0 0 0 0 0 0 0 0 0 0 0 # 0 0 0 0 0 0 0 0 0 0 0 0 0
0 # 0 0 0 0 0 0 0 0 0 # # 0 0 0 0 0 0 0 0 0 # 0 0 0 0 0 0 0 0 0 0 0 0
0 # 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 # 0 0 0 0 0 # 0 0 0 0 0 0 0 0 0 0 0
0 # # 0 0 0 0 0 0 # 0 0 0 # 0 0 0 0 0 0 0 0 0 # 0 0 0 0 0 0 0 0 0 0
0 0 # 0 # 0 0 0 # 0 0 0 0 0 0 0 0 0 0 0 0 0 0 # 0 # 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 # 0 0 0 0 0 # 0 0 0 # 0 0 0 0 0 0 0 0 # 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 # # 0 0 0 0 0 0 0 0 0 0 0 # 0 0 0 0 0 # # 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 # 0 0 # 0 # 0 0 0 0 0 0 # 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 # 0 # 0 0 0 0 0 0 # 0 # 0 0 0 0 # 0 0 0 0 0
0 0 0 0 # # 0 0 0 0 # 0 0 0 0 0 0 0 0 0 0 # 0 0 0 0 0 0 0 0 # 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 # # 0 0 0 # # 0 0 0 0 0 0 # 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 # 0 0 # 0 # 0 0 0 0 0 0 0 # # 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 # 0 # 0 0 0 # 0 # 0 0 0 0 # 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 # 0 0 0 0

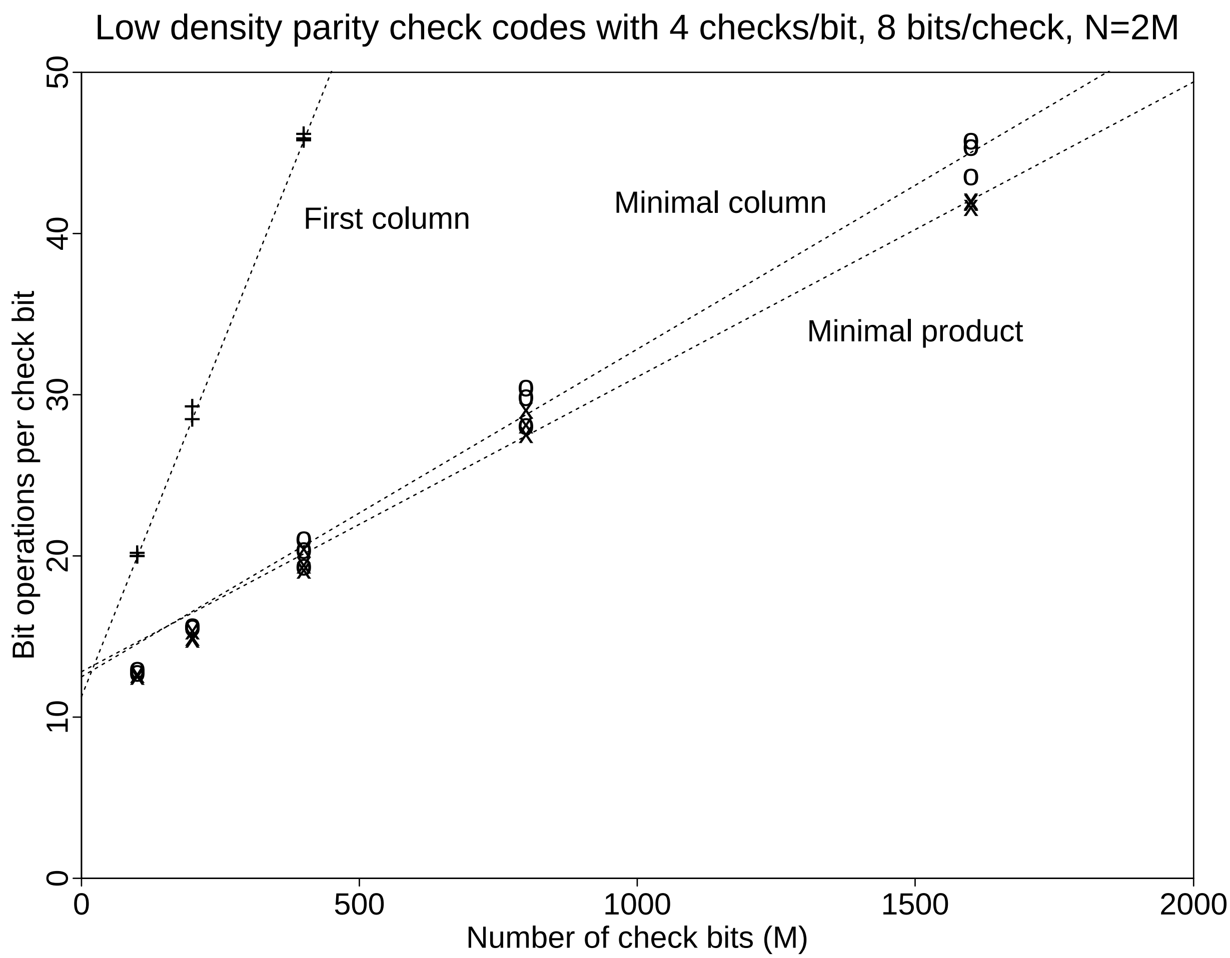
The **U** *Matrix for This Code,*
As Found With the Minimal Product Heuristic

#	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	#	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	#	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	#	0	0	0	0	0	0	0	0	0
0	0	0	#	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	#	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	#
0	0	0	0	0	#	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	#	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	#	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	#	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	#	0	0	#	0	0	0	0	0
0	0	0	0	0	0	0	0	#	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	#	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	#	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	#	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	#	0
0	0	0	0	0	0	0	0	0	0	0	#	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	#	0	0	0	0	0	0	0	0	0	#	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	#	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	#	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	#	0	0	0	0	0	0	0	0	0	#	0	0	0	0	0	0	0	0	#	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	#	0	0	0	0	0	0	0	0	0	0	0	#	0	0	#	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	#	0	0	0	0	0	0	0	0	0	0	0	#	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	#	0	0	0	0	0	0	0	0	0	0	0	0	0	0	#	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	#	0	0	0	0	#	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	#	0	0	0	0	#	0	0	0	0	0	0	#	0	#	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	#	0	#	0	0	0	0	#	#	0	#	#	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	#	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	#	0	0	0	0	#	0	#	0	0	0	0	#
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	#	0	0	0	0	0	0	0	0	#	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	#	#	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	#	#	#	0	0	0	0	#	0	#
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	#	0	0	#	#	0	#	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	#	0	0	0	#	#	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	#	0	0	#	0	#	#
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	#	0	0	0	0	#
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	#	0	0	#	#
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	#	#	#	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	#	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	#	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	#

Results on Codes With 3 Checks per Bit



Results on Codes With 4 Checks per Bit



Summary

- A fairly standard LU decomposition approach can greatly reduce the number of bit operations for encoding low density parity check codes.

- For standard LDPC codes, the number of operations per check still grows linearly with block size, but at a slow rate. Hence encoding still takes time proportional to N^2, but with a small constant factor.

- For moderate block sizes, dense matrix operations can still be faster, especially in software, due to the parallelism possible by operating on 32 bits at a time.

- The process of forward substitution resembles that of encoding a recursive convolutional code.