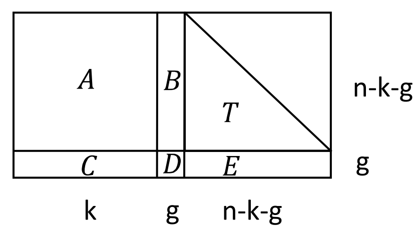
Low-Complexity Encoding Algorithm for LDPC Codes

Problem:
Given the following matrix (imagine a larger matrix with a small number of ones)

$$H = \begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix},$$

and the vector of information bits $s = (1\ 0\ 1\ 0\ 1\ 0)$, find a codeword $x' \in GF(2)^{12}$ satisfying $Hx'^T = 0$ and corresponding to the information bits s. In other words, show how to encode information bits s into a valid codeword. Utilize the fact that the matrix H is sparse to decrease the complexity.



Solution: The encoding procedure for LDPC codes consists of 2 parts: the preprocessing part (done only once per matrix H) and the encoding of given information bits.

In the preprocessing part, we need to find row and column permutations of the matrix H, such that after these permutations, the matrix can be written in the form

$$\widehat{H} = \begin{pmatrix} A & B & T \\ C & D & E \end{pmatrix},$$

where T is a lower triangular matrix with ones on the diagonal and the matrix $F = ET^{-1}B$ is of a full rank. The matrices are of the following size: $A \in GF(2)^{n-k-g \times k}$, $B \in GF(2)^{n-k-g \times g}$, $C \in GF(2)^{g \times k}$, $D \in GF(2)^{g \times g}$, $E \in GF(2)^{g \times n-k-g}$, $T \in GF(2)^{n-k-g \times n-k-g}$. Calculate F^{-1} (this will be a dense matrix, but is of a very small size). This operation needs to be done only once. See "Part 1" below for an example.

For a given information bit vector s, the encoding part finds a valid codeword $x' \in GF(2)^{12}$, $Hx'^T = 0$ corresponding to s. Let σ be the column permutation obtained in step 1, then we set $\sigma(x') = x = (s\ x_p)$, where x_p is the vector of parity-check bits. This vector is unknown in the beginning and will be obtained from the equation $\widehat{H}\left(s\ x_p\right)^T = 0$. See "Part 2" below for an example of how to find the parity-check bits.

Part 1: (Approximate triangulation of sparse H)

The preprocessing part consists of finding row and column permutations of the matrix H, such that the right part of the resulting matrix is almost lower triangular. Finding the best (with the smallest possible gap g) row and column permutations is a hard problem in general. We will use the following simple suboptimal greedy algorithm that will give us permutations with a sufficiently small g with a small complexity.

Algorithm for approximate triangulation:

[INITIALIZE]: Start with $H_{n-k} = H$, $t = n - k$, and $g = 0$. Define the <u>residual degree of a column</u> in H_t as the number of ones in the first t rows. Go to CONTINUE.

[CONTINUE]: If $\mathrm{t} = 0$ then stop and output $\widehat{\mathrm{H}} = \mathrm{H}_0$. Otherwise, if the minimum positive residual degree in H_t is 1 go to EXTEND, else go to CHOOSE.

[EXTEND]: Choose a random column c of residual degree 1 in $\mathrm{H_t}$. Let r be the row (in the range $[1, \mathrm{t}]$) of H_t that contains the non-zero entry in column c. Swap column c with column $t + k + g$ and row r with row t. Call the resulting matrix H_{t-1}. Decrease t by one and go to CONTINUE.

[CHOOSE]: Choose a random column c with the minimal POSITIVE residual degree, call the degree d. Let $r_1, \dots, r_d$ be the rows of H_t (in the range $[1, t]$) which contain the d residual non-zero entries in column c. Swap column c with column $t + k + g$. Swap row r_1 with row t and move rows $r_2, \dots, r_d$ to the bottom of the matrix. Call the resulting matrix H_{t-d}. Decrease t by d, increase g by $d - 1$ and go to CONTINUE.

We run the algorithm on the matrix H and describe all the steps. We start with $H_6 = H$, $t = 6$, and $g = 0$. By row r in matrix H_t, we mean the r-th row in this matrix. The same holds for columns.

H6	1	2	3	4	5	6	7	8	9	10	11	12
1	0	1	0	1	0	0	1	1	1	0	1	0
2	0	1	1	0	0	0	1	1	0	1	0	1
3	1	0	0	0	1	1	0	1	0	0	1	1
4	0	1	1	0	1	0	1	0	1	1	0	0
5	0	0	0	1	1	1	0	0	1	1	1	0
6	1	0	1	1	0	1	0	0	0	0	0	1

The minimum positive residual degree of H_6 is $d = 2$ (first column, $c = 1$). From the CHOOSE step, $r_1 = 3$ and $r_2 = 6$. We swap rows 3<->5 (row 6 is on the bottom already) and columns 1<->12 obtaining H_4 and $t = 4$, $g = 1$:

H4	12	2	3	4	5	6	7	8	9	10	11	1
1	0	1	0	1	0	0	1	1	1	0	1	0
2	1	1	1	0	0	0	1	1	0	1	0	0
5	0	0	0	1	1	1	0	0	1	1	1	0
4	0	1	1	0	1	0	1	0	1	1	0	0
3	1	0	0	0	1	1	0	1	0	0	1	1
6	1	0	1	1	0	1	0	0	0	0	0	1

The minimum POSITIVE residual degree of H_4 is $d = 1$ (first column, $c = 1$). From the EXTEND step, $r = 2$. We swap rows (relative numbers in H_4) 2<->4 and columns 1<->11 obtaining H_3 and $t = 3$, $g = 1$:

H3	11	2	3	4	5	6	7	8	9	10	12	1
1	1	1	0	1	0	0	1	1	1	0	0	0
4	0	1	1	0	1	0	1	0	1	1	0	0
5	1	0	0	1	1	1	0	0	1	1	0	0
2	0	1	1	0	0	0	1	1	0	1	1	0
3	1	0	0	0	1	1	0	1	0	0	1	1
6	0	0	1	1	0	1	0	0	0	0	1	1

The minimum POSITIVE residual degree of H_3 is $d = 1$ (third column, $c = 3$). From the EXTEND step, $r = 2$. We swap rows (relative numbers in H_3) 2<->3 and columns 3<->10 obtaining H_2 and $t = 2$, $g = 1$:

H2	11	2	10	4	5	6	7	8	9	3	12	1
1	1	1	0	1	0	0	1	1	1	0	0	0
5	1	0	1	1	1	1	0	0	1	0	0	0
4	0	1	1	0	1	0	1	0	1	1	0	0
2	0	1	1	0	0	0	1	1	0	1	1	0
3	1	0	0	0	1	1	0	1	0	0	1	1
6	0	0	0	1	0	1	0	0	0	1	1	1

The minimum POSITIVE residual degree of H_2 is $d = 1$ (second column, $c = 2$). From the EXTEND step, $r = 1$. We swap rows (relative numbers in H_2) 1<->2 and columns 2<->9 obtaining H_1 and $t = 1$, $g = 1$:

H1	11	9	10	4	5	6	7	8	2	3	12	1
5	1	1	1	1	1	1	0	0	0	0	0	0
1	1	1	0	1	0	0	1	1	1	0	0	0
4	0	1	1	0	1	0	1	0	1	1	0	0
2	0	0	1	0	0	0	1	1	1	1	1	0
3	1	0	0	0	1	1	0	1	0	0	1	1
6	0	0	0	1	0	1	0	0	0	1	1	1

Finally, the minimum POSITIVE residual degree of H_1 is $d = 1$ (first column, $c = 1$). From the EXTEND step, $r = 1$. We columns 1<->8 obtaining the result $\widehat{H} = H_0$. We found row and column permutations with the gap $g = 1$.

H0	8	9	10	4	5	6	7	11	2	3	12	1
5	0	1	1	1	1	1	0	1	0	0	0	0
1	1	1	0	1	0	0	1	1	1	0	0	0
4	0	1	1	0	1	0	1	0	1	1	0	0
2	1	0	1	0	0	0	1	0	1	1	1	0
3	1	0	0	0	1	1	0	1	0	0	1	1
6	0	0	0	1	0	1	0	0	0	1	1	1

Make sure that matrix $F = D - ET^{-1}B$ is of full rank (we will need this in Part 2).
Solution: $D = (0)$, $B = (0\ 1\ 1\ 1\ 0)^T$, $E = (0\ 0\ 1\ 1\ 1)$, $T^{-1}B = (0\ 1\ 0\ 0\ 0)^T$, $ET^{-1}B = 0$. Matrix $F = (0)$ and thus is NOT of rank 1. This can be corrected by swapping 7th with i-th column in H_0, where $1 \leq i \leq 6$ (do not touch 8th and other columns since they are in the required form). Swap column 3<->7 and calculate F again. This step is ALWAYS possible iff the original matrix H has full rank. Explain why?

H	8	9	7	4	5	6	10	11	2	3	12	1
5	0	1	0	1	1	1	1	1	0	0	0	0
1	1	1	1	1	0	0	0	1	1	0	0	0
4	0	1	1	0	1	0	1	0	1	1	0	0
2	1	0	1	0	0	0	1	0	1	1	1	0
3	1	0	0	0	1	1	0	1	0	0	1	1
6	0	0	0	1	0	1	0	0	0	1	1	1

Now $D = (0)$, $B = (1\ 0\ 1\ 1\ 0)^T$, $E = (0\ 0\ 1\ 1\ 1)$, $T^{-1}B = (1\ 1\ 0\ 0\ 1)^T$, $ET^{-1}B = 1$ and $F = D - ET^{-1}B = 0 - 1 = 1$ and thus F has rank 1 as desired.

If $g > 1$, consider the following approach of how to find F having a full rank. Let $\mathcal{B}$ be the set of columns participating in the matrix B. First, realize which column in $\mathcal{B}$ is redundant by removing it from F and recalculating the rank of F. If the rank stays the same, the column is redundant. Remove all redundant columns from $\mathcal{B}$. Let $\mathcal{A}$ be the set of columns participating in matrix A. Consider which column from $\mathcal{A}$ is not redundant in $\mathcal{B}$ by placing it to $\mathcal{B}$ and calculating the rank of F. If the rank increases, keep it in $\mathcal{B}$.

Part 2: (Finding a parity-check bit vector)

Let $\widehat{H}$ be in the form

$$\widehat{H} = \begin{pmatrix} A & B & T \\ C & D & E \end{pmatrix},$$

where all matrices are SPARSE (they were obtained by column and row permutations of the sparse matrix H). Calculate

$$H' = \begin{pmatrix} A & B & T \\ C & D & E \end{pmatrix} \begin{pmatrix} I & 0 \\ -ET^{-1} & I \end{pmatrix} = \begin{pmatrix} A & B & T \\ C - ET^{-1}A & D - ET^{-1}B & 0 \end{pmatrix}.$$

From Part 1, we know that $F = D - ET^{-1}B$ has full rank.

Questions for you:

- How would you interpret the matrix multiplication step in the context of a Gaussian elimination algorithm?
- Convince yourself that $\widehat{H}\left(s\ x_p\right)^T = 0$ iff $H'\left(s\ x_p\right)^T = 0$.

Write $x_p = \left(x_{p1}\ x_{p2}\right)$, where x_{p1} contains the first g parity-check bits and x_{p2} contains the remaining $n - k - g$ bits. We need to solve

$$\begin{pmatrix} A & B & T \\ C - ET^{-1}A & D - ET^{-1}B & 0 \end{pmatrix} \left(s\ \ x_{p1}\ x_{p2}\right)^T = 0$$

for an unknown parity-check vector $x_p = (x_{p1}\ x_{p2})$. Since we know s, we may simplify the system to

$$\begin{pmatrix} B & T \\ F & 0 \end{pmatrix} \left(x_{p1}\ x_{p2}\right)^T = \begin{pmatrix} z \\ w \end{pmatrix},$$

where $z = As^T$, $w = (C - ET^{-1}A)s^T$ and $F = D - ET^{-1}B$ is a full-rank matrix of size $g \times g$ (g is very small when compared to n). The rank property comes from Part 1. Matrix F is not sparse in general.

Since we have F^{-1}, we can find $x_{p1} = F^{-1}w$ and then $x_{p2} = T^{-1}(z - Bx_{p1}^T)$.

Complexity of the described algorithm and other implementation issues:

Let G, H be SPARSE matrices of a size such that the product $u = GHv$ is defined for some vector v (v does not need to be sparse). We have two ways how to calculate u, either $u = (GH)v$ or $u = G(Hv)$. In the first case we need to calculate matrix*matrix product which is much more complex than matrix*vector product (and the result does not need to be sparse again). From this reason, always use the second approach to evaluate any term similar to $u = GHv$. Unfortunately, Matlab uses the first approach if you do not specify the order by brackets, thus use u=G*(H*v) notation.

All operations except $x_{p1} = F^{-1}w$ can be done in linear time by using the order of evaluation mentioned above. This is because all matrices except F^{-1} are sparse. Any term of the type $u = T^{-1}v$ can be calculated by backsubstitution in linear time. For example the term $w = (C - ET^{-1}A)s^T$ should be evaluated as

$$w = (Cs^T) - \left(E\left(T^{-1}(As^T)\right)\right).$$

In Matlab notation:

```
w = C*s'-E*(T\(A*s'));
```

Solution to the numerical example:

$$\widehat{H} = \left(\begin{array}{cccccc|c|ccccc} 0 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 1 \\ \hline 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \end{array}\right),$$

$$A = \begin{pmatrix} 0 & 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 \end{pmatrix}, B = \begin{pmatrix} 1 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}, C = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \end{pmatrix}^T, D = (0), E = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 1 \\ 1 \end{pmatrix}^T, T = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 \end{pmatrix}, F = (1)$$

$s = (1\ 0\ 1\ 0\ 1\ 0)$, $z = As^T = (1\ 0\ 0\ 0\ 0)^T$, $w = 0$, $F^{-1} = (1)$, $x_{p1} = w = 0$, $x_{p2} = (1\ 1\ 1\ 0\ 1)^T$

$$x = \left(s\ x_{p1}\ x_{p2}\right) = (1\ 0\ 1\ 0\ 1\ 0, 0\ , 1\ 1\ 1\ 0\ 1)$$

From Part 1, we have the column permutation $\sigma = (8, 9, 7, 4, 5, 6, 10, 11, 2, 3, 12, 1)$. Its inverse is $\sigma^{-1} = (12,\ 9, 10,\ 4,\ 5,\ 6,\ 3,\ 1,\ 2,\ 7,\ 8,\ 11)$.

```
% calculate inverse permutation in Matlab
sigma = [8 9 7 4 5 6 10 11 2 3 12 1];
sigma_inv(sigma) = 1:12;
```

Finally, we need to apply the inverse permutation to obtain the codeword

$$x' = \sigma^{-1}(x) = (1\ 1\ 1\ 0\ 1\ 0\ 1\ 1\ 0\ 0\ 1\ 0).$$

Now $Hx' = 0$ as desired.