Low-density parity-check codes

Last modified: 2026-11-25 01:48:21 (PST)

A linear code is described by a parity-check matrix. When it has few ones, each check involves only a few positions of the word.

Sparse parity checks

Definition 0.1 (Regular low-density parity-check code)**.** Let $j, k \geq 1$ be integers. A (j, k)**-regular LDPC code** (low-density parity-check code) consists of the words $y \in \mathbb{F}_2^n$ with $Hy = 0$, where H has j ones in every column and k ones in every row, and zeros elsewhere. The rows of H need not be independent. When no row is a sum of other rows, H is a parity-check matrix of this linear code.

Example 0.1 (A $(2, 3)$-regular code of length 6)**.**
Take four items, labeled $1, 2, 3, 4$, and the six pairs of them. Let H have a row for each item and a column for each pair, with a 1 where the item lies in the pair. Every pair holds two items and every item lies in three pairs, so H has two ones in each column and three in each row, and it gives a $(2, 3)$-regular LDPC code of length 6. The R chunk below builds H, checks the counts, and counts the words y with $Hy = 0$ modulo 2.

```
pairs <- t(combn(4, 2))
H <- sapply(seq_len(nrow(pairs)), function(e) as.integer(1:4 %in% pairs[e, ]))
H
#>      [,1] [,2] [,3] [,4] [,5] [,6]
#> [1,]    1    1    1    0    0    0
#> [2,]    1    0    0    1    1    0
#> [3,]    0    1    0    1    0    1
#> [4,]    0    0    1    0    1    1
colSums(H)
#> [1] 2 2 2 2 2 2
rowSums(H)
#> [1] 3 3 3 3
words <- as.matrix(expand.grid(rep(list(0:1), 6)))
is_codeword <- apply(words, 1, function(y) all((H %*% y) %% 2 == 0))
sum(is_codeword)
#> [1] 8
```

There are 8 codewords, so the dimension is $\log_2$ of that number, which is 3.

Theorem 0.1 (The rate of a regular LDPC code)**.** *Let C be a (j, k)-regular LDPC code of length n. Then the rate of C is at least $1 - j/k$.*

Proof

Proof. Let H be as in the definition of C, with m rows and n columns. Count the ones of H in two ways. Each of the n columns holds j ones, so H has nj ones. Each of the m rows holds k ones, so H has mk ones. So $nj = mk$, and $m = nj/k$.
The codewords are the words y with $Hy = 0$, so their number is 2^{n-r}, where r is the rank of H. The rank of H is at most its number of rows, so $r \leq m$, and C has at least 2^{n-m} codewords. Its rate is $(n - r)/n$, which is at least $(n - m)/n = 1 - j/k$. □

Example 0.2 (The rate of the length-6 code)**.**
For the $(2, 3)$-regular LDPC code of Example 0.1, Theorem 0.1 gives a rate of at least $1 - 2/3$. It has 8 codewords, so its rate is $\log_2(8)/6$.

```
1 - 2 / 3
#> [1] 0.3333333
log2(8) / 6
#> [1] 0.5
```

The rate is 1/2, which is larger than 1/3. The four rows of H add to zero modulo 2, because every column has two ones, so H has rank 3, not 4.

Remark 0.1 (Where LDPC codes come from). Gallager (1962) introduced low-density parity-check codes, and MacKay (2003, chap. 47) covers them under the name Gallager codes, including decoding by message-passing algorithms [Section 47.2].

Decoding

Definition 0.2 (Tanner graph)**.** Let H be a parity-check matrix with m rows and n columns. The **Tanner graph** of H has a *variable node* for each column of H and a *check node* for each row of H, with an edge between variable node i and check node a exactly when $H_{ai} = 1$. Every edge joins a variable node to a check node.

Example 0.3 (The Tanner graph of the length-6 code)**.**
For the H of Example 0.1, the variable nodes are the six pairs of items, and the check nodes are the four items. Each pair is joined to the two items it contains, so every variable node has two edges. Each item is joined to the three pairs that contain it, so every check node has three edges. The R chunk below lists the check nodes joined to each variable node.

```
apply(H, 2, function(column) which(column == 1))
#>      [,1] [,2] [,3] [,4] [,5] [,6]
#> [1,]    1    1    1    2    2    3
#> [2,]    2    3    4    3    4    4
```

Definition 0.3 (Bit-flipping decoder)**.** Let H be a parity-check matrix, and let $y \in \mathbb{F}_2^n$ be a received word. A check node a is *unsatisfied* when the ath entry of Hy is 1. The **bit-flipping decoder** repeats the following step:

- Send each check node's status to the variable nodes joined to it in the Tanner graph.
- Count, for each variable node, the unsatisfied check nodes joined to it.
- Flip the entry of y at a variable node with the largest count, taking the smallest index if several tie.

It stops after a fixed number of iterations, or when no check node is unsatisfied.

Example 0.4 (Correcting one flipped entry of the length-6 code)**.**
Send the all-zero word, which is a codeword of the LDPC code of Example 0.1, and suppose the channel flips the first entry. The R chunk below runs the bit-flipping decoder and prints the received word and the number of unsatisfied checks before each iteration.

```
y <- c(1, 0, 0, 0, 0, 0)
for (step in 1:3) {
  syndrome <- as.vector(H %*% y) %% 2
  counts <- as.vector(t(H) %*% syndrome)
  cat("y =", y, "| unsatisfied checks =", sum(syndrome), "\n")
  if (all(syndrome == 0)) break
  flip <- which.max(counts)
  y[flip] <- 1 - y[flip]
}
#> y = 1 0 0 0 0 0 | unsatisfied checks = 2
#> y = 0 0 0 0 0 0 | unsatisfied checks = 0
```

The flipped entry lies in two checks, and both are unsatisfied, so its count is 2, while every other entry has a count of at most 1. The decoder flips the first entry, which restores the sent word.

References

Gallager, Robert G. 1962. "Low-Density Parity-Check Codes." *IEEE Transactions on Information Theory* 8 (1): 21–28. https://doi.org/10.1109/TIT.1962.1057683.

MacKay, David J. C. 2003. *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press.