Low-Density Parity-Check codes

Martin Leslie

Department of Mathematics
University of Arizona

October 21, 2009

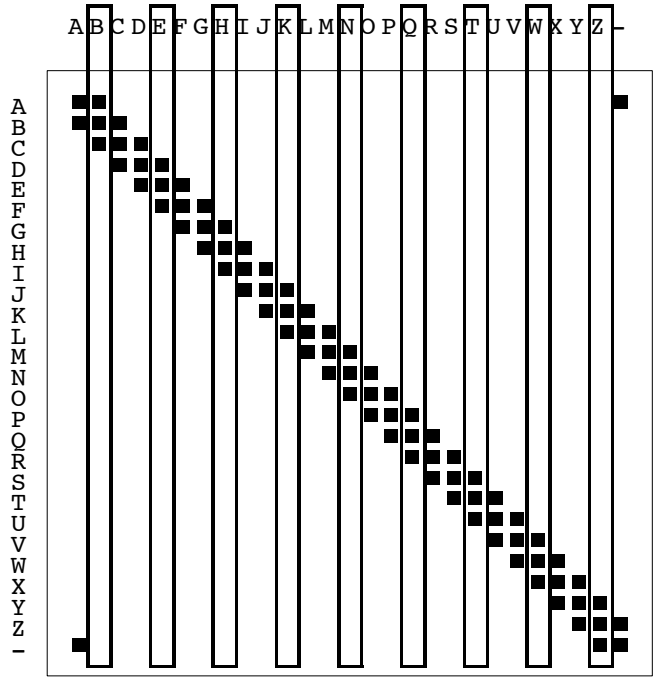
A toy model

- Our first communication channel is the *noisy typewriter*.
- We can send the letters A, B, ..., Y, Z, _.
- But the typewriter has some problems: If B is sent, receive one of A, B or C, each with probability 1/3.
- The same is true for the other letters (wrapping around the 27th letter _).
- Can you come up with a communication strategy that lowers the error rate? Can you make the error rate approach or equal zero? What does your strategy do to the information rate?

An optimal solution

- If we use only 9 of the letters, leaving gaps of two between each, then there can be no confusion as to what was sent.
- So only ever send B, E, H, ..., Z.
- Of course we want to send other letters so we need some encoding of the 'information' letters into our 'code' letters.
- Notice that $27^2 = 9^3$ so we can encode blocks of 2 information letters into blocks of 3 code letters.
- Say AA $\rightarrow$ BBB, AB $\rightarrow$ BBE, AC $\rightarrow$ BBH, ...

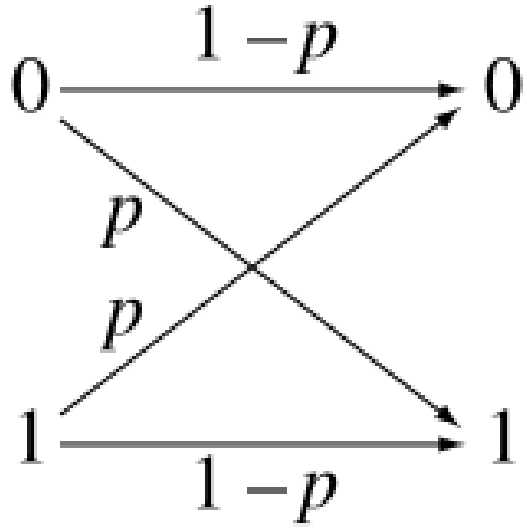
- The picture:



- The *information rate* for this code is 2/3 information letters per code letter. The *error rate* is zero.

A more realistic channel

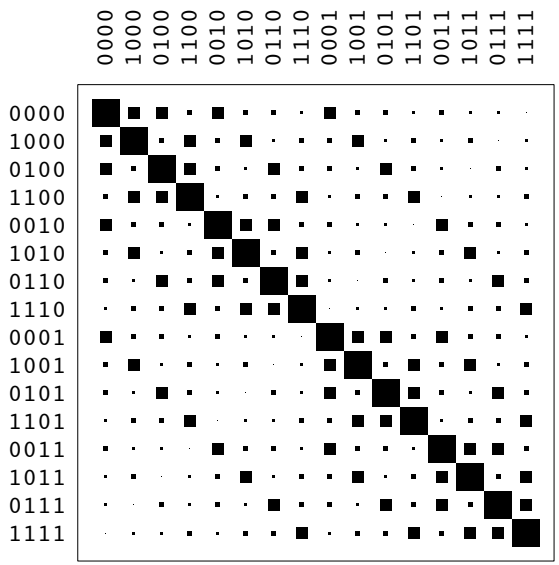
- The channel model we will use is the binary symmetric channel (BSC) which takes a binary input and with probability $p < 1/2$ switches it.



- This is a good model for deep space communications but not so good for hard drives or for terrestrial communications where errors often come in bursts.
- Can we find a good communication strategy for this channel?

A handwaving answer

- The same kind of thinking as for the noisy typewriter works, at least approximately.
- The picture:



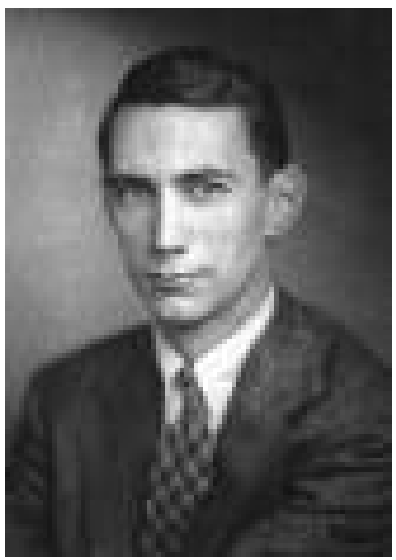
$N = 4$

- Consider a block code where we choose some number of inputs of length N that have almost non-overlapping outputs.

Noisy–channel coding theorem

Theorem (Shannon, 1948)

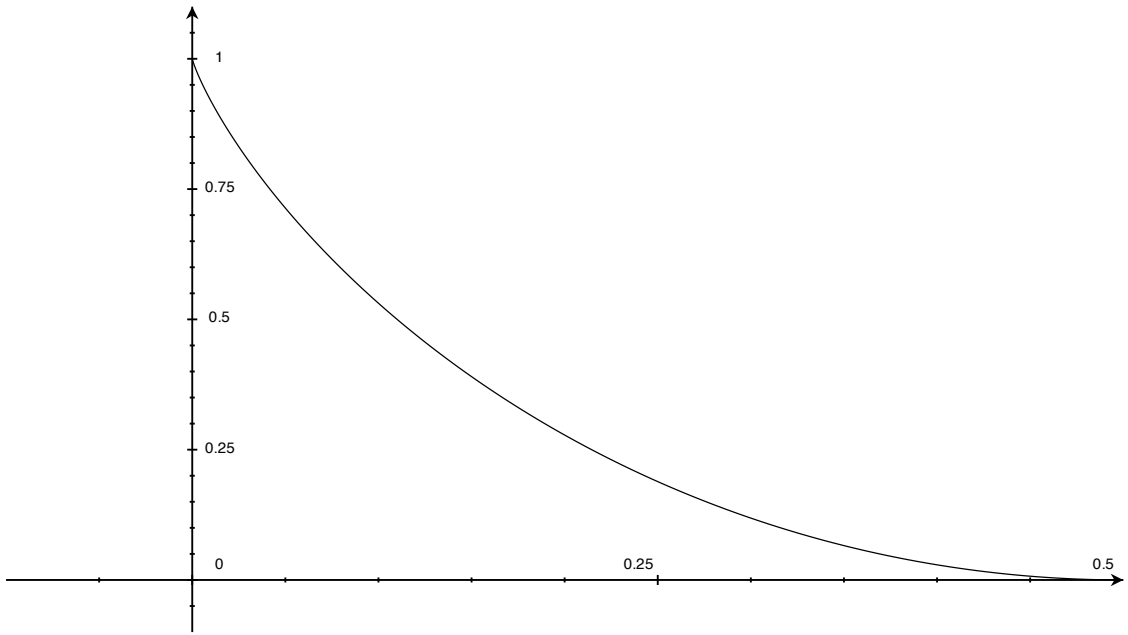
A given channel has a capacity C. Fix an $R < C$. Then for all $\epsilon > 0$ there exists a sufficiently large n and a code with information rate R, block length n and probability of decoding a block incorrectly less than ϵ.



Capacity

- The capacity of the noisy typewriter is $2/3$.
- The capacity of the binary symmetric channel with crossover probability p is

$$C = 1 + p\log_2(p) + (1-p)\log_2(1-p).$$



Linear codes

- An $[n, k]_2$ code C is a k-dimensional linear subspace of $\mathbb{F}_2^n$.
- The basis vectors of C are the rows of the *generating matrix* G.
- With this matrix we can carry out encoding by the function from $\mathbb{F}_2^k \to \mathbb{F}_2^n$ that sends $u \mapsto uG$.
- Then the information rate is $R = k/n$.

The Hamming [7,4] code

- Let $G = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{pmatrix}$
- Then the codewords of C are

0000 000	1000 110	0010 111
1111 111	0100 011	1001 011
	1010 001	1100 101
	1101 000	1110 010
	0110 100	0111 001
	0011 010	1011 100
	0001 101	0101 110

The Parity Check Matrix

- If a code C has generating matrix $G = (I \ P)$ then define its *parity check matrix* to be $H^T = \begin{pmatrix} P \\ I \end{pmatrix}$.
- Notice that $GH^T = P + P = 0$.
- So if $c \in C$ we know $c = uG$ and thus

$$cH^T = uGH^T = 0.$$

- For the Hamming [7,4] code we have $H^T = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$

Hamming distance

- The Hamming distance $d_H(u, v)$ for $u, v \in \mathbb{F}_2^n$ is the number of places in which u and v differ.
- This satisfies all the axioms of a metric on $\mathbb{F}_2^n$.
- The Hamming weight is the number of 1's in a vector,

$$\mathrm{wt}(u) = d_H(u, 0).$$

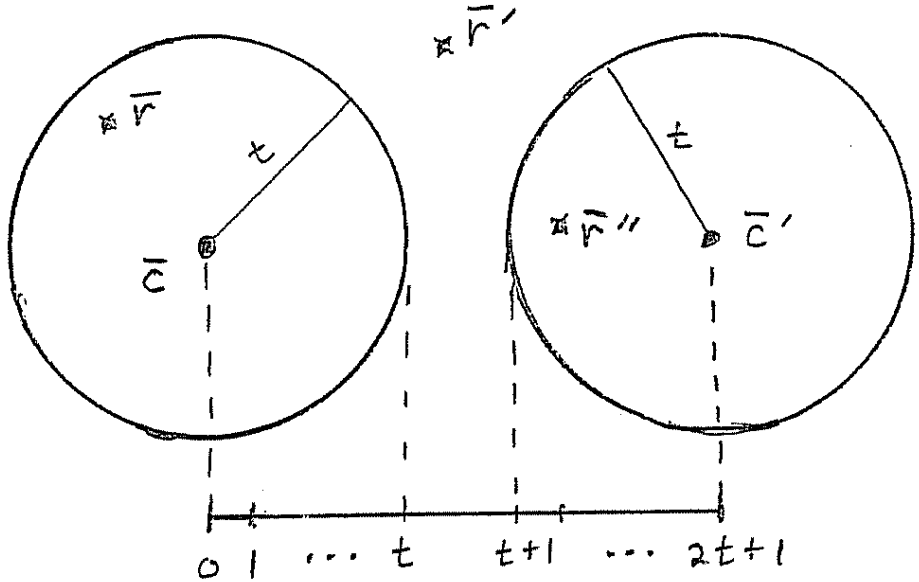
- Define d to be the minimum Hamming distance between any two vectors in C. Then

$$d = \min_{u \neq v \in C} d_H(u, v) = \min_{u \in C \setminus \{0\}} \mathrm{wt}(u).$$

- Then we talk about an $[n, k, d]_2$ code.

Nearest neighbour decoding

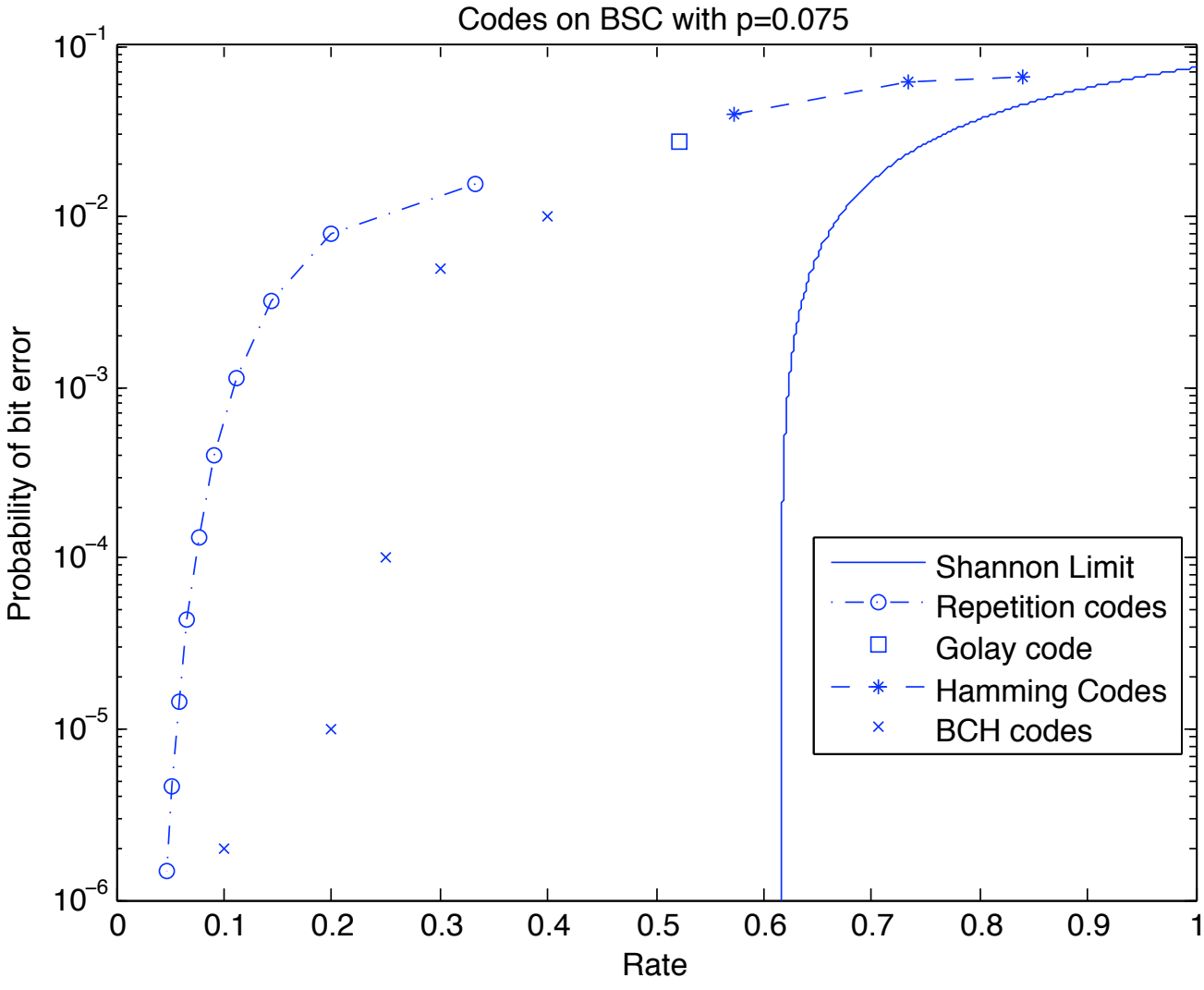
- If we receive $u \in \mathbb{F}_2^n$ we decode it to an element v of C for which $d_H(u, v)$ is minimum.
- It's possible that this is incorrect decoding but it is certainly the best choice on average.
- If d is the minimum distance of C then we can correct at least $t = \left\lfloor \frac{d-1}{2} \right\rfloor$ errors.



The search for good codes

- Shannon's proof of his noisy channel coding theorem uses random codes. These are not practical.
- Since that time, coding theorists have searched for codes that approach the capacity of the channel and have practical encoding and decoding algorithms.
- One major idea was algebraic codes: Hamming codes, Reed–Solomon codes, Golay codes, BCH codes...

- None of these are much good:

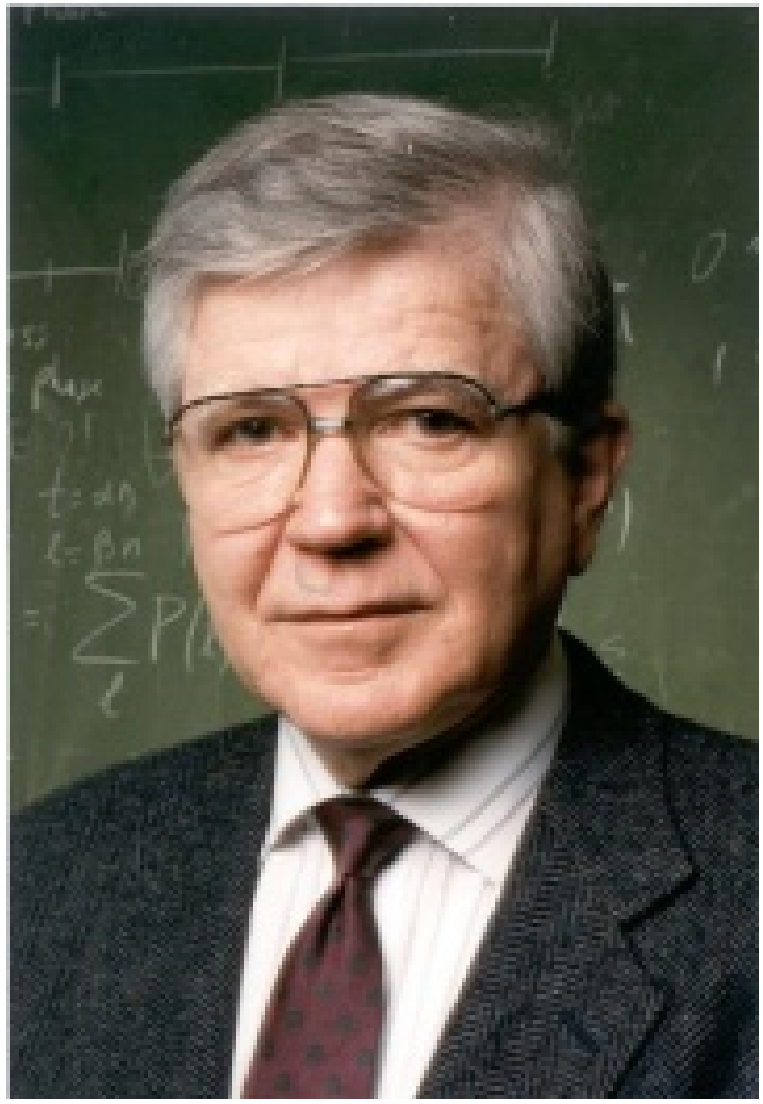


The big idea

- Create random codes and then come up with a practical idea to decode them.
- If you create a random large parity check matrix with a low density of 1's then on average it will generate a good code.
- For these parity check matrices there exists a non-optimal, but good enough, decoder that is practical.
- These Low Density Parity Check (LDPC) codes were discovered in 1996 by MacKay and Neal.

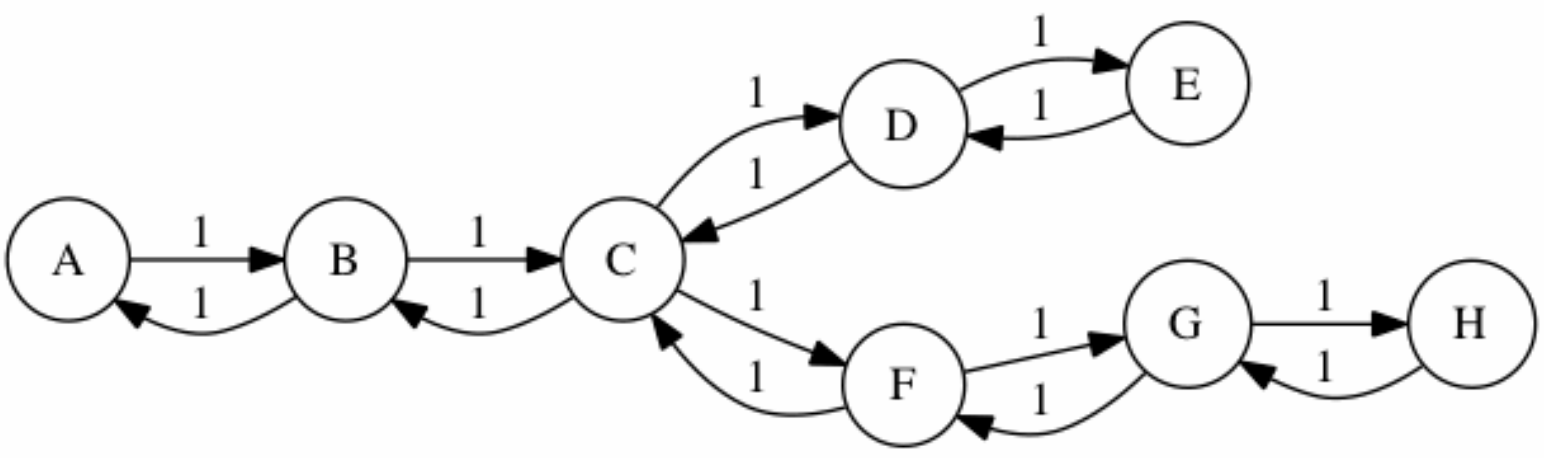
Well sort of...

- Actually, Robert Gallager had invented LDPC codes in his 1960 thesis.



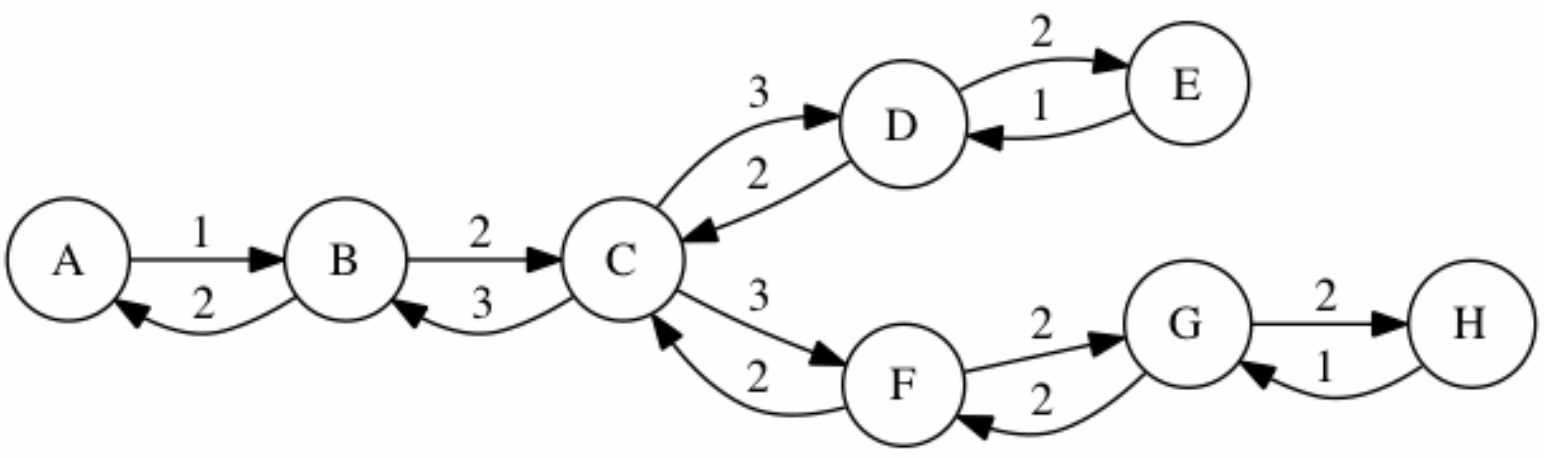
Message passing

- A distributed way to count soldiers:
  - At time step 1, each soldier says 1 to each soldier next to them.
  - Each time step after this, soldier i says to soldier j, the sum of all numbers said to him by soldiers except j plus 1 for himself.
  - When this stabilizes, at each soldier, the sum of the inputs to that soldier plus 1 should be the number of soldiers.



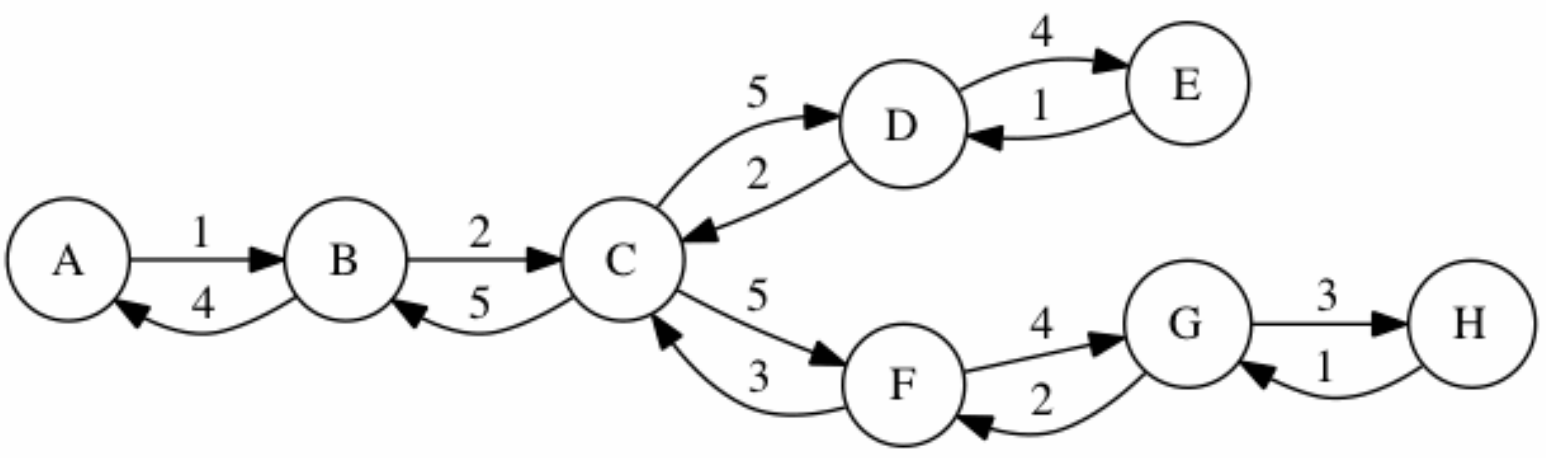
Message passing

- A distributed way to count soldiers:
  - At time step 1, each soldier says 1 to each soldier next to them.
  - Each time step after this, soldier i says to soldier j, the sum of all numbers said to him by soldiers except j plus 1 for himself.
  - When this stabilizes, at each soldier, the sum of the inputs to that soldier plus 1 should be the number of soldiers.



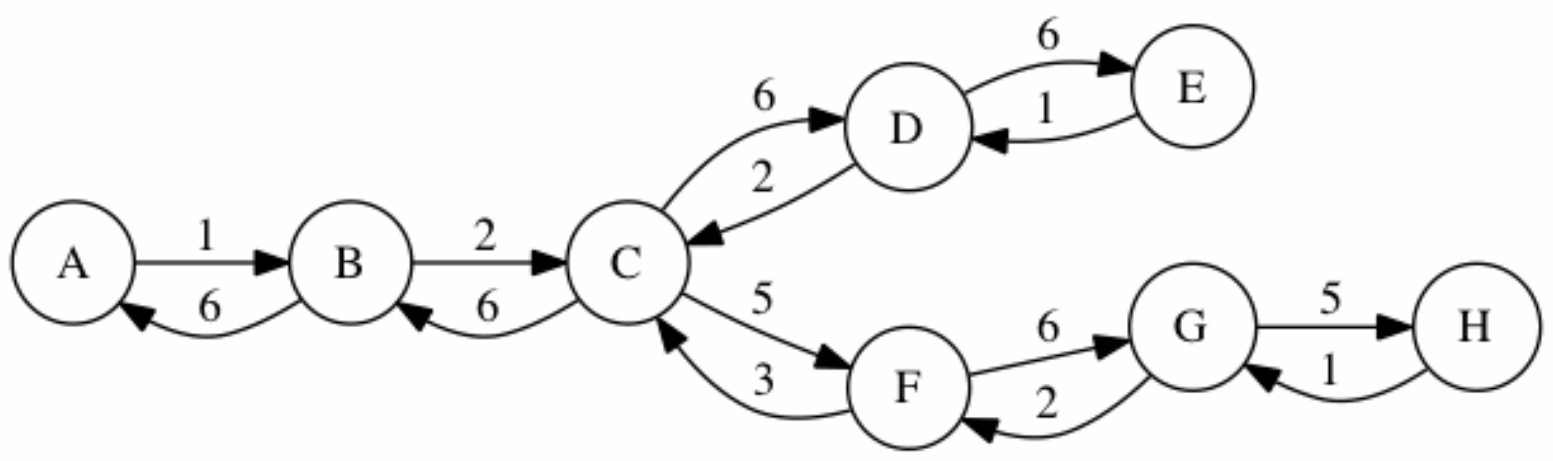
Message passing

- A distributed way to count soldiers:
  - At time step 1, each soldier says 1 to each soldier next to them.
  - Each time step after this, soldier i says to soldier j, the sum of all numbers said to him by soldiers except j plus 1 for himself.
  - When this stabilizes, at each soldier, the sum of the inputs to that soldier plus 1 should be the number of soldiers.



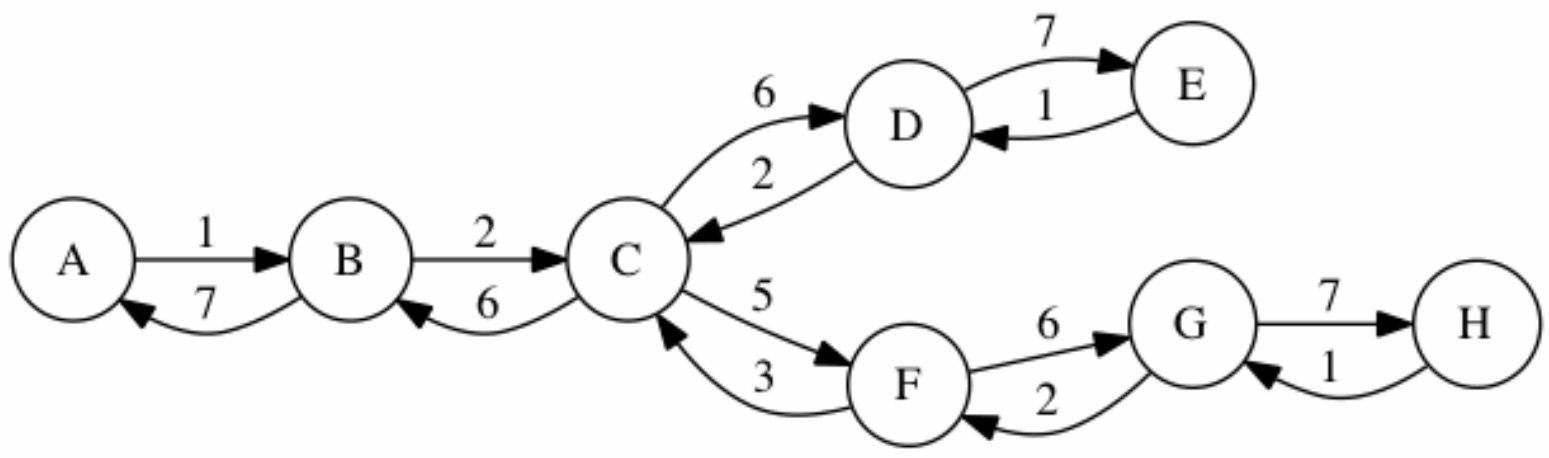
Message passing

- A distributed way to count soldiers:
  - At time step 1, each soldier says 1 to each soldier next to them.
  - Each time step after this, soldier i says to soldier j, the sum of all numbers said to him by soldiers except j plus 1 for himself.
  - When this stabilizes, at each soldier, the sum of the inputs to that soldier plus 1 should be the number of soldiers.



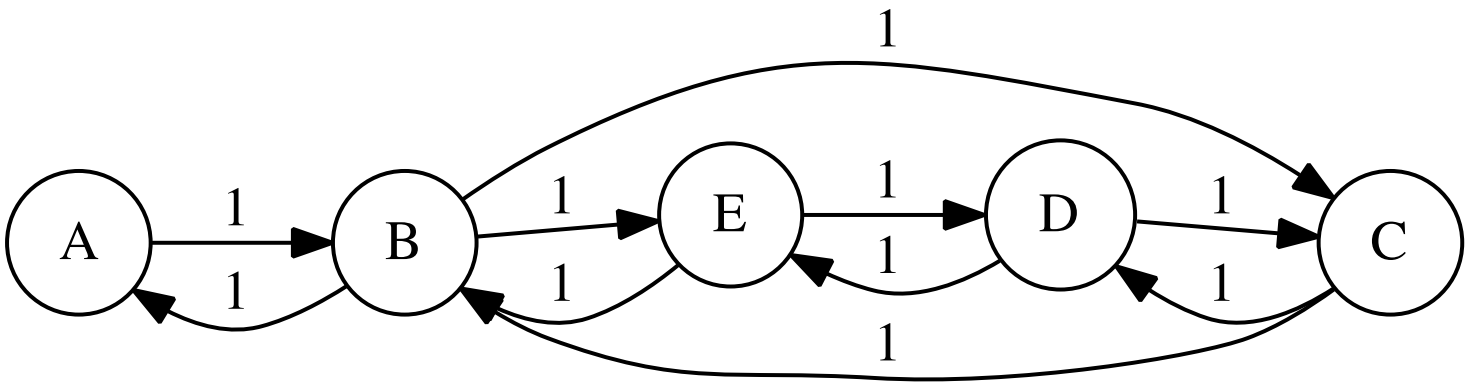
Message passing

- A distributed way to count soldiers:
  - At time step 1, each soldier says 1 to each soldier next to them.
  - Each time step after this, soldier i says to soldier j, the sum of all numbers said to him by soldiers except j plus 1 for himself.
  - When this stabilizes, at each soldier, the sum of the inputs to that soldier plus 1 should be the number of soldiers.



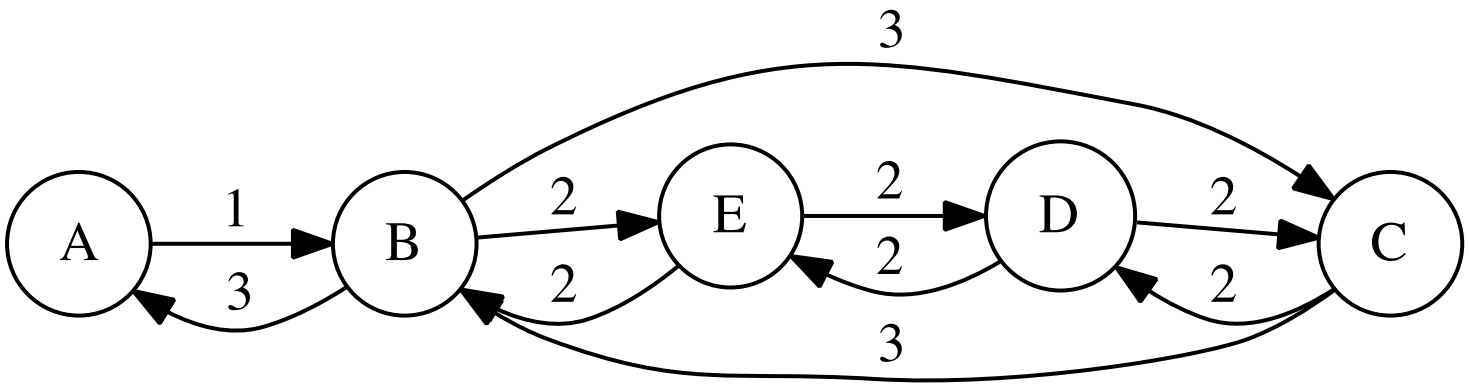
Cycles are bad

- When we have cycles in our soldier graph:



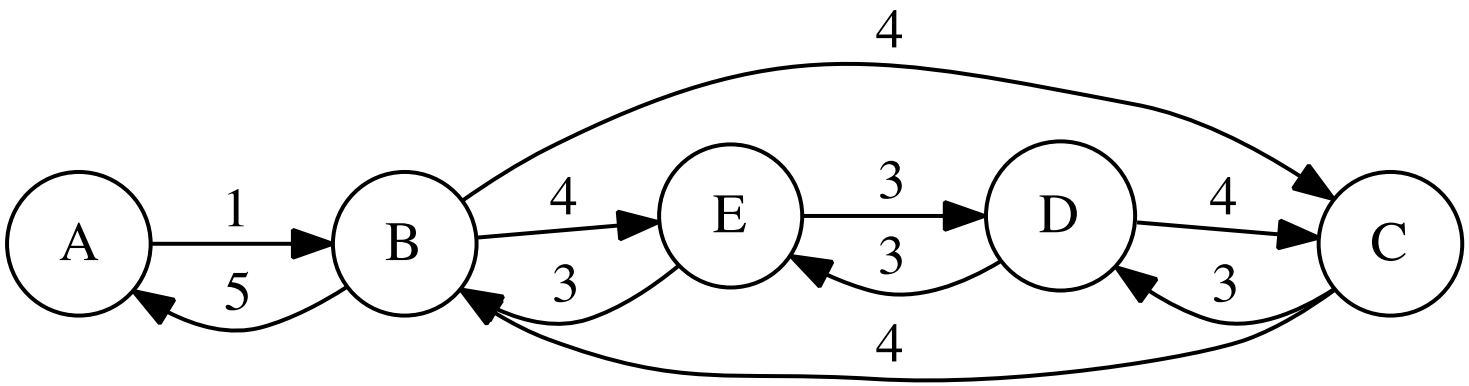
Cycles are bad

- When we have cycles in our soldier graph:



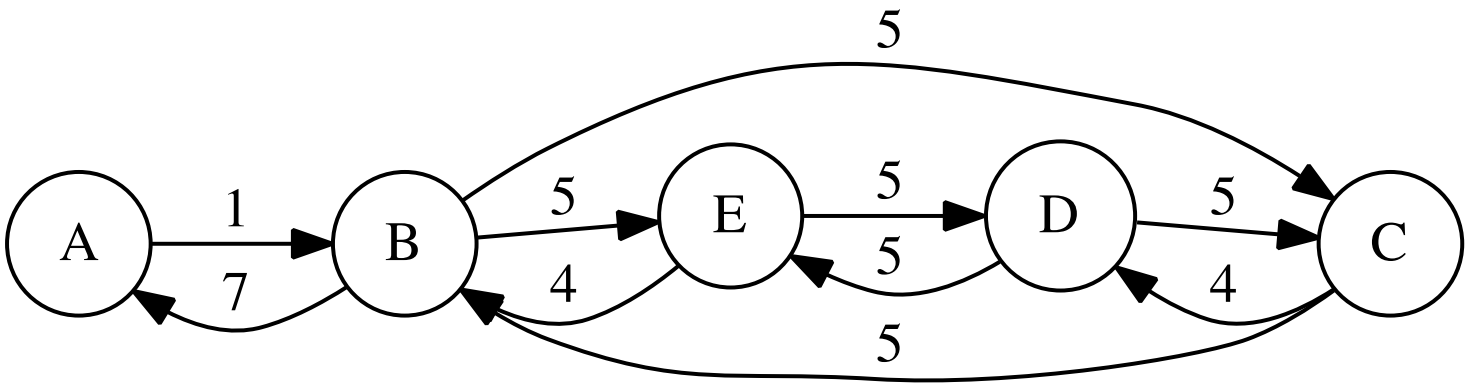
Cycles are bad

- When we have cycles in our soldier graph:



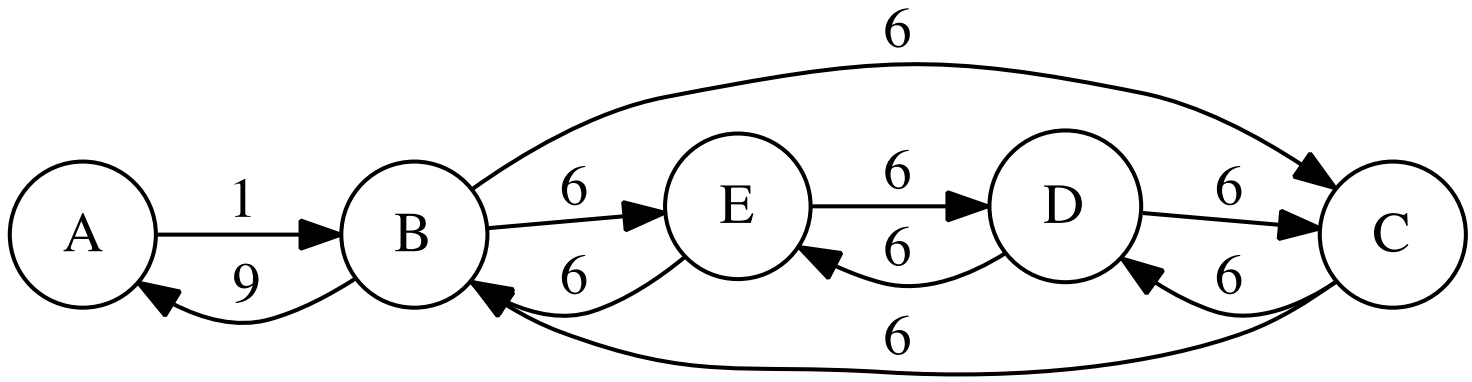
Cycles are bad

- When we have cycles in our soldier graph:



Cycles are bad

- When we have cycles in our soldier graph:

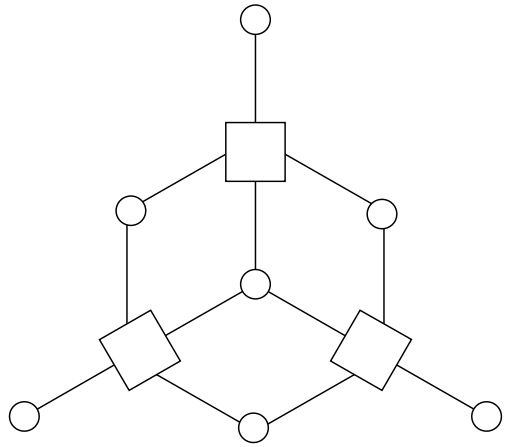


How to view a code as a graph

- Recall our parity check matrix for the Hamming [7,4] code:

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

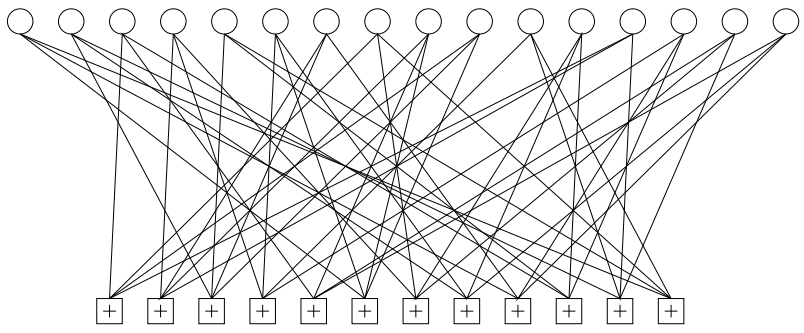
- We can think of it as a graph with 7 variable nodes (circles) and 3 check nodes (squares)



But we need it bigger

- For example, an H of size 12×16:

$$\mathbf{H} = \begin{bmatrix}
0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \\
0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\
1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\
0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\
1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\
1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\
0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 \\
0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\
1 & 0 & 1 & 0 & 1 & 0 & 1 & 0
\end{bmatrix}$$



- An LDPC code that approaches capacity might have an H of size 10000×20000.

The decoder

- The decoder is an iterative belief propagation algorithm.
- Each variable node has a probability that represents the belief that the algorithm has that that node should be a 1 in the final decoding.
- The check nodes can take these beliefs about the codeword and see if they are consistent.
- The algorithm goes up and down the graph, communicating and updating these beliefs, until the probability of decoding error is below some threshold.
- If the graph corresponding to the parity check matrix had no cycles, this decoder would be provably optimal.
- If we have only large cycles then hopefully it's good enough.

More details: a naive implementation

- We use ratios of belief, $\frac{P(x_i=0)}{P(x_i=1)}$.

More details: a naive implementation

- We use ratios of belief, $\frac{P(x_i=0)}{P(x_i=1)}$.
- Step 0: The variable nodes set their initial belief ratio to $(1-p)/p$ or $p/(1-p)$ after receiving a 0 or 1 respectively. Then they send that number to each check node they are connected to.

More details: a naive implementation

- We use ratios of belief, $\frac{P(x_i=0)}{P(x_i=1)}$.
- Step 0: The variable nodes set their initial belief ratio to $(1-p)/p$ or $p/(1-p)$ after receiving a 0 or 1 respectively. Then they send that number to each check node they are connected to.
- Step 1: Each check node sends a message to each variable node it is connected to which gives the check node's belief ratio for that variable node. This is calculated by the check node based on the messages received from other variable nodes.

More details: a naive implementation

- We use ratios of belief, $\frac{P(x_i=0)}{P(x_i=1)}$.
- Step 0: The variable nodes set their initial belief ratio to $(1-p)/p$ or $p/(1-p)$ after receiving a 0 or 1 respectively. Then they send that number to each check node they are connected to.
- Step 1: Each check node sends a message to each variable node it is connected to which gives the check node's belief ratio for that variable node. This is calculated by the check node based on the messages received from other variable nodes.
- Step 2: Each variable node takes the belief ratios from the check nodes it is connected to and multiplies them. This is its new belief ratio.

More details: a naive implementation

- We use ratios of belief, $\frac{P(x_i=0)}{P(x_i=1)}$.
- Step 0: The variable nodes set their initial belief ratio to $(1-p)/p$ or $p/(1-p)$ after receiving a 0 or 1 respectively. Then they send that number to each check node they are connected to.
- Step 1: Each check node sends a message to each variable node it is connected to which gives the check node's belief ratio for that variable node. This is calculated by the check node based on the messages received from other variable nodes.
- Step 2: Each variable node takes the belief ratios from the check nodes it is connected to and multiplies them. This is its new belief ratio.
- Step 3: Create a possible decoding. If it's not a codeword go back to Step 1.

The Final Picture